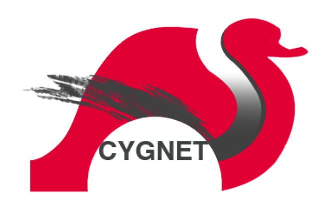






# Cygnet

# Bachelorarbeit




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuender Dozent:** Prof. Dr. Andreas Steffen

**Betreuer:** Tobias Brunner

**Gegenleser:** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# Abstract


## Ausgangslage

Jeder Informatikbetreiber muss sich heute mit der "Bring Your Own Device" (BYOD) Thematik befassen, da die Mitarbeiter mit ihren privaten Notebooks, Tablets oder Smartphones auf die Netzwerkdienste ihrer Organisation zugreifen möchten. Dies birgt Risiken, da diese Geräte nur teilweise kontrolliert werden können. Es können aufgrund der vielen unterschiedlichen Geräte und Client-Betriebssysteme nur sehr schwer einheitliche Richtlinien für die Benutzung der Dienste erstellt werden. Durch den gleichzeitigen privaten Einsatz der Geräte kann es zu unfreiwilligen Datenfreigaben ("Leaks") kommen. Malware, die sich auf Client-Geräten während dem privaten Gebrauch einnistet, kann sich während der geschäftlichen Verwendung im Firmennetzwerk verbreiten. Die strongSwan VPN-Lösung versucht mit Implementierung des Trusted Network Connect Standards eine Milderung dieses Problems zu erreichen. Die vorliegende Arbeit soll eine Möglichkeit bieten, innerhalb einer BYOD-Umgebung eine möglichst einheitliche Richtlinie für alle Geräte zu konfigurieren, die von strongSwan durchgesetzt werden kann.

## Technologie

Cygnet ist als web-basierte Applikation realisiert, um die Richtlinien zu definieren. Diese speichert Daten in einer SQLite-Datenbank, welche gleichzeitig zum Datenaustausch mit strongSwan dient. Für die Webapplikation wurde auf das Python-Framework Django gesetzt in Kombination mit JQuery für clientseitigen Code. Die Applikation ist für den Einsatz auf einem Apache-Webserver gedacht. Bei der Umsetzung wurde auf eine möglichst generische Implementierung geachtet. Die von strongSwan eingesetzten Integrity Measurement Verifier sind weiterhin Änderungen unterworfen und auch die Liste von möglichen Richtlinien ist noch unvollständig. Cygnet sollte dieser Dynamik gerecht werden. Es wurde eine Schnittstelle zu StrongSwan definiert und implementiert, um Arbeitsschritte zwischen Cygnet und strongSwan zu übermitteln und Resultate auszuwerten. Dazu gehört eine Webapplikation, um die Richtlinien zu konfigurieren.

## Ergebnis

Als Ergebnis liegt eine leicht bedienbare, ansprechende Applikation vor, die einem Administrator flexible Möglichkeiten bietet, eine umfassende Sicherheitsrichtlinie für die Clients seines Netzwerks zu konfigurieren und durchzusetzen.

# Ausblick

Die vorliegende Arbeit hatte zum Ziel, eine Erweiterung zu strongSwan zu sein, daher existiert eine gewisse Kopplung zwischen den beiden Produkten. Es wäre in Zukunft denkbar, die Schnittstelle noch generischer zu spezifizieren und in Kollaboration mit der Trusted Computing Group als Zusatz zu TNC zu definieren, so dass auch andere IPSec-Implementierungen mit Cygnet zusammenarbeiten könnten.

# Erklärung der Eigenständigkeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder was mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben und,
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil-Jona, den 13. Juni 2013

Name, Unterschrift

Stefan Rohner

Name, Unterschrift

Marco Tanner

# Inhaltsverzeichnis

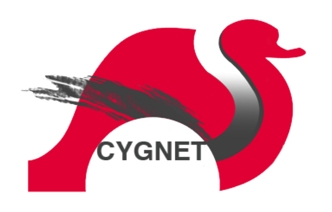






# Cygnet

# Requirements




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser:** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 1. Einleitung

## 1.1. Zweck

Dieses Dokument erläutert die spezifischen Anforderungen, welche aus der Aufgabenstellung und Gesprächen mit Herrn Steffen für die geplante Applikation “Cygnet” erarbeitet wurden.

## 1.2. Allgemeine Beschreibung

### 1.2.1. Produktfunktion

Die Applikation soll die Möglichkeit bieten, mit einer grafischen Oberfläche Policy-Regeln zu administrieren, welche ein Whitelisting wie auch ein Blacklisting von Geräten und deren Software-Paketen ermöglichen. Die Policies sollen in einer Datenbank gespeichert werden und Gesundheitstests für Android-Geräte und die zugehörigen Resultate werden ebenfalls darin erfasst.

### 1.2.2. Benutzercharakteristik

Als Benutzer kommen in erster Linie Netzwerkadministratoren in Frage, die ein StrongSwan IPSec Netz betreiben. Es ist davon auszugehen, dass die Benutzer bereits Erfahrung im Umgang mit Computern und Web-Administrations-Oberflächen gesammelt haben.

### 1.2.3. Abhängigkeiten

Für die Erfassung von potentiell gefährlichen Packages wird eine bereits bestehende Datenbank verwendet und erweitert. Ausserdem wird die StrongSwan IMV und somit auch die implementierte Erweiterung der StrongSwan - Android App genutzt. Sie ermöglicht die Validierung von Endgeräten.

# 2. Ausgangslage

## 2.1. Szenario

Die Firma Examplis beschäftigt 50 Mitarbeiter. Davon sind 25 Aussendienstmitarbeiter, 3 IT-Administratoren, 10 Softwareentwickler und der Geschäftsleiter mit Android-Smartphones ausgestattet. Die Aussendienst-MA's sind zusätzlich mit einem Android-Tablet unterwegs.
Es existiert ein IPSec - VPN Netzwerk, das den Mitarbeitern auch von unterwegs oder vom Home-Office aus Zugriff auf die internen Netzwerkressourcen ermöglicht.
Die Netzwerkadministratoren möchten sicherstellen, dass alle Android-Clients den Firmenanforderungen entsprechen und keine Sicherheitsrisiken mit sich bringen.
Sie stellen eine Reihe von Anforderungen auf, die jedes Gerät erfüllen muss, bevor es ins VPN-Netzwerk eingelassen wird:

- Die Aussendienstmitarbeiter dürfen keine Server-Software installiert haben, es sollen keine Listening Sockets vorhanden sein. Software-Entwickler sind von dieser Einschränkung nicht betroffen, da sie einen SSH-Server für Ihre Arbeit benötigen.
- Um Sicherheitsrisiken zu vermeiden, soll sichergestellt werden, dass alle Mitarbeiter die neuesten Updates für Ihre Apps installieren. Software-Entwickler sind teilweise auf bestimmte Versionen Ihrer Apps angewiesen, aber auch sie müssen zumindest die neuesten Sicherheitsupdates installiert haben.
- Wenn Software Updates fehlen, soll es den Benutzern doch möglich sein, auf das Internet zuzugreifen, um die Updates nachträglich zu installieren. Der Zugriff auf Firmenressourcen soll aber verwehrt bleiben.
- Einige Software-Entwickler haben ihr Gerät spasseshalber "gerootet". Diese Geräte sollen im Netzwerk nicht erlaubt sein.
- Gemäss einer Informatik-Zeitschrift kursiert zur Zeit ein Android-Rootkit, das einen Key-Logger in die "bash" integriert und die Logs auf einen externen Server speichert. Deshalb soll mittels Datei-Hash überprüft werden, ob die "bash"-Kopie des Users infiziert wurde.

## 2.2. Persona IT-Administrator



Abbildung 2.1.: Freddy Firewall

Freddy Firewall[1] ist Netzwerkadministrator der Examplis, er ist 36 Jahre alt und hat eine Lehre als Informatiker Fachrichtung Systemtechnik absolviert. Er beschäftigt sich zu 70% mit der firmeninternen Netz-Infrastruktur, die restliche Zeit betreut er das hauseigene ERP-System. Er arbeitet mit einem Desktop-Computer, kann aber bei Bedarf von zuhause aus via VPN auf Serverlogs zugreifen.

Freddy bevorzugt für seine Arbeit grafische Tools, da er sich so komplexe Systemaufbauten besser vorstellen kann. Besonders bei der Konfiguration von Firewalls und/oder Switches verlässt er sich lieber auf das mitgelieferte Web-UI, als sich kryptische Kommandozeilenbefehle zu merken.
Seine grösste Sorge ist eine Attacke durch Hacker, welche die Kundendaten oder den Quellcode der von der Examplis entwickelten Programme kopieren könnten. Es stört ihn daher sehr, dass er keine Kontrolle hat, welche Apps seine Benutzer auf ihren Smartphones und Tablets installieren. Insbesondere probiert auch der Geschäftsleiter in seiner Freizeit gerne verschiedenste Games und andere Apps aus, die teilweise versuchen, ihn zu In-App-Käufen zu bewegen.

[1] Bild: http://public-domain-photos.com

# 3. Use Cases

## 3.1. Aktoren & Stakeholder

| Aktor, Stakeholder | Interessen |
|---|---|
| System (Cygnet) | Möchte jederzeit korrekte, konsistente Daten beinhalten. |
| Administrator | Möchte Clients seines strongSwan-Netzwerks möglichst einfach verwalten. |
| strongSwan IMV | Möchte Logins von strongSwan-Clients und die zugehörigen Sicherheitschecks abwickeln. |
| Android-Benutzer | Möchte sich möglichst schnell und jederzeit in das Netz einwählen. |

Tabelle 3.1.: Aktoren & Stakeholder

## 3.2. UC0: Login strongSwan-Client

Ein strongSwan IMV beginnt mit einem Verbindungsaufbau mit einem Client. Mittels der Android-ID kann er sich die auszuführenden Prüfungen für das Gerät bei Cygnet abholen. Er führt die Checks durch und schreibt deren Resultate in die passende Cygnet-Tabelle. Anhand dieser Ergebnisse kann entschieden werden, ob das Gerät ins Netz eingelassen wird oder nicht. Wenn der Login-Vorgang abgeschlossen ist, meldet dies der IMV bei Cygnet. Cygnet aktualisiert dann anhand der Resultate die Historie des Geräts und die Policy-Bestimmungen für zukünftige Login-Versuche.

## 3.3. UC1: Einrichten einer Policy

Der Administrator möchte ein Regelwerk aufstellen, welche Android-Geräte seiner Benutzer sich zu welchen Konditionen ins Netz einwählen dürfen.

### 3.3.1. UC1.1: Device- und Gruppenmanagement

Alle Geräte, die sich bereits einmal via strongSwan ins Netz eingewählt haben, sind automatisch im System erfasst. Der Administrator teilt die Geräte in logische Gruppen ein, die Gruppenstruktur orientiert sich an der Firmenhierarchie oder den unterschiedlichen Sicherheitsstufen, die für Geräte gelten können.

### 3.3.2. UC1.2: Erstellen einer Policy

Der Administrator definiert gemäss seinen Sicherheitsanforderungen Policiy-Objekte. Er legt fest, was die Policy prüfen soll und welche Reaktion ausgelöst wird, wenn die Policy nicht erfüllt wird (z.B. Wird das Gerät blockiert).

### 3.3.3. UC1.3: Erzwingen von Policies

Die definierten Policies werden den bestehenden Gruppen zugeordnet: Jede Gruppe kann beliebig viele Policies aktiviert haben. Zusätzlich definiert der Administrator, wie oft eine Prüfung durchgeführt werden soll.

## 3.4. UC2: Logs prüfen

Ein Benutzer meldet, er könne sich nicht mehr im VPN anmelden. Der Administrator öffnet deshalb Cygnet und kontrolliert die Ergebnisse der letzten Login-Versuche. Er erkennt, welche Policy zur Blockierung des Geräts geführt hat und bietet dem Benutzer Hilfestellung beim Erfüllen der Policy-Bedingungen.

# 4. Funktionale Anforderungen

## 4.1. Muss-Kriterien

### 4.1.1. M0: CRUD

Cygnet soll das bestehende Datenbankschema der “ipsec.config.db” abbilden, erweitern und die gängigen CRUD-Operationen für den Benutzer in einer Webapplikation bereitstellen.

### 4.1.2. M1: Policy Management

Die bestehende Datenbank muss so erweitert werden, dass Policies für Geräte und Gerätegruppen festgelegt werden können. Die Policies bestimmen, welche Sicherheitsprüfungen zu welchem Zeitpunkt ausgeführt werden. Auch diese Funktionen sollen in Cygnet verwaltbar sein.

### 4.1.3. M2: strongSwan Logins

Die effektiv geltenden Policies für ein Gerät sollen als “Worklist” in einer separatenen, aggregierten Tabelle abgelegt werden, damit die strongSwan-IMVs einfach darauf zugreifen können. Die Resultate der einzelnen Prüfungen sollen durch den IMV in der Datenbank gespeichert werden. Nach dem bearbeiten Login-Vorgang kann der IMV eine Neuberechnung der Worklist für das Gerät auslösen.

## 4.2. Soll-Kriterien

### 4.2.1. S0: Authentifizierung mittels Passwort

Um sicher zu stellen, dass nur befugte Personen Policies und Einstellungen bearbeiten, wird die Web-Applikation nur mit gültigem Benutzernamen und Passwort zugänglich sein. Als optionale Erweiterung kann ein “Read-only” Zugriff auf gewisse Bereiche möglich sein.

### 4.2.2. S1: REST-ful URLs

Die URLs von cygnet sollen nach dem Representational State Transfer Paradigma gestaltet werden, z.B. “`http://domain.com/devices/\{Device-ID\}/details`”.

### 4.2.3. S2: Einfache Analyse für Administrator

Wenn ein Kunde/Client sich nicht beim Netzwerk einloggen kann, soll es für den Administrator einfach ersichtlich sein, welche Policies auf den Client wirken und welche davon für den Misserfolg verantwortlich ist, damit dem Kunden in kurzer Zeit geholfen werden kann.

### 4.2.4. S3: Internationalisierung

Die Applikation soll internationalisiert sein, alle Meldungen und Beschriftungen sollen in austauschbaren “Resource-Files” gespeichert sein. Die Lokalisierung kann so zu einem späteren Zeitpunkt einfach nachgeholt werden.

## 4.3. Kann-Kriterien

### 4.3.1. K0: Benutzerunterstützung

So weit möglich, unterstützt Cygnet die Aktionen des Benutzers. Das heisst, dass vom Benutzer auszufüllendende Felder mit sinnvollen Standardwerten befüllt sein sollen und dass die Eingaben des Benutzers nach Möglichkeit automatisch ergänzt werden (Autocompletion).

### 4.3.2. K1: JSON API

Damit zu einem späteren Zeitpunkt weitere Anwendungen oder Interfaces an die cygnet-DB entwickelt werden können, soll eine JSON-API entwickelt werden, damit Daten auch programmatisch ausgelesen und evtl. auch geschrieben werden können.

### 4.3.3. K2: Ubuntu Clients

Es soll Unterstützung für Ubuntu-strongSwan Clients (und Derivate wie Xubuntu und Kubuntu) implementiert werden. Für diese Geräte sind einige Rahmenbedingungen anders als bei Android-Geräten. Beispielsweise muss die eindeutige Identifizierung ohne Android-ID gelöst werden und auch die Policies müssen weiter anpassbar sein, da es zum Beispiel durchaus üblich ist, dass Server-Applikationen auf Ubuntu installiert sind (z.B. sshd).

### 4.3.4. K3: Mobile Version

Das Web-UI soll auch auf mobilen Geräten korrekt, resp. optimiert dargestellt werden. Die Funktionalität soll so weit möglich komplett mit mobilen Geräten bedient werden können.

### 4.3.5. K4: Statistiken

Es können Statistiken über die Datenbank wie zum Beispiel Erfolgsquote bei Login-Versuchen über Zeit betrachtet werden.

# 5. Nichtfunktionale Anforderungen

## 5.1. Leistungsanforderung

Das verwendete System hat zwei Hauptaufgaben: Erstens müssen für jeden VPN- bzw. strongSwan Login-Vorgang die entsprechenden Resultate ausgewertet und die für das Gerät geltenden Policy-Informationen aktualisiert werden. Dies muss in einem Zeitrahmen geschehen, der den Login-Prozess für das Android-Gerät nicht merklich verlangsamt. Die Verarbeitung eines Logins sollte daher maximal zwei Sekunden dauern.
Die zweite Hauptaufgabe besteht im Bereitstellen einer herkömmlichen webbasierten Applikation zur Verwaltung der Datenbank. Die Antwort-Zeiten sollten genügend klein sein, dass es sich für den Benutzer “responsive” anfühlt. Längere Operationen sollten mit einer Art Fortschrittsanzeige visualisiert werden, damit der Benutzer nicht das Gefühl bekommt, die Applikation reagiere nicht mehr auf seine Eingaben.

## 5.2. Mengenanforderung

Für ein grosses VPN-Netzwerk können tausende Geräte einen Zugang haben oder bereits mit dem Netzwerk verbunden sein. Die Anzahl *gleichzeitiger* Logins dürfte sich aber selbst bei einem solchen Netzwerk im Rahmen von 1 bis maximal 5 bewegen. Diese müssen gemäss den oben beschriebenen Leistungsanforderungen abgefertigt werden können.

Die Datenmenge hängt davon ab, wieviele Geräte und Softwarepakete verwaltet bzw. kontrolliert werden. Die Anzahl Einträge pro Tabelle wird aber kleiner als 100’000 sein, was bei weniger als 20 Tabellen selbst für SQLite als kleinere Datenbank gewertet werden kann.

## 5.3. Qualitätsanforderungen

Die geplante Software soll nach der ISO-Norm 9126 entwickelt werden. [1]
Es werden die folgenden Qualitätsmerkmale definiert und umgesetzt:

### 5.3.1. Funktionalität

#### Angemessenheit

Die Applikation soll möglichst alle der genannten Anforderungen erfüllen. Allfällige weitere Funktionen können hinzugefügt werden, falls der Zeitplan es zulässt. Dennoch soll die Applikation und die Datenbank schlank und übersichtlich bleiben.

#### Richtigkeit

Sämtliche Inhalte der Datenbank sollen jederzeit konsistent und richtig sein. Datenbanktransaktionen sollen das ACID-Prinzip erfüllen. Informationen über Software-Pakete werden von einer externen Quelle eingeholt, von deren Korrektheit ausgegangen wird.

#### Interoperabilität

Die Applikation soll reibungsfrei mit der bestehende strongSwan-Applikation zusammenarbeiten. Die Daten, welche mit strongSwan ausgetauscht werden, sollen dasselbe Format haben wie es derzeit implementiert ist.

#### Sicherheit

Die Verbindung zur Webapplikation soll TLS-gesichert stattfinden, damit die Kommunikation mit den Clients verschlüsselt abläuft. Die Datenbank selbst ist nicht verschlüsselt.
Der Zugriff auf die Webapplikation soll grobgranular über Benutzer bzw. Benutzergruppen gesteuert werden können. Es gibt mindestens einen Administrationsbenutzer und einen Readonly-Benutzer.

### 5.3.2. Zuverlässigkeit

#### Reife

Die Applikation soll strongSwan-Logins zuverlässig und fehlerfrei abwickeln. Alle Verwaltungsoperationen sollen mit dem geplanten Webinterface möglich sein. Wenn der Cygnet-Service nicht in Betrieb ist, sollen trotzdem strongSwan-Logins möglich bleiben.

[1] Es wurde bewusst Norm 9126 gewählt, obwohl sie durch ISO-25000 abgelöst wurde. ISO-9126 ist weniger umfangreich, aber dem Projektumfang angemessen.

#### Fehlertoleranz

Die Webapplikation soll Fehler vom Benutzer nach Möglichkeit erkennen, Eingaben werden auf ihr Format hin geprüft. Der Benutzer wird bei Fehlern visuell darüber informiert. Sollte eine Transaktion fehlschlagen oder ein Default-Wert eingesetzt werden, soll stets die konservative, sicherere Variante gewählt werden.

### 5.3.3. Wiederherstellbarkeit

Ein Programmabsturz soll die Datenbank in einem konsistenten Zustand belassen. Ein Backup der Datenbank kann mit einer simplen File-Copy-Operation erstellt werden und ebenso einfach zurückgespielt werden.

### 5.3.4. Benutzbarkeit

#### Verständlichkeit

Das Webinterface soll für einen erfahrenen Benutzer ohne Lesen des Manuals verständlich und intuitiv bedienbar sein. Für weitere Informationen wird eine Bedienungsanleitung erstellt.

### 5.3.5. Effizienz

#### Zeitverhalten

Siehe 5.1 Leistungsanforderung (Seite 17).

#### Verbrauchsverhalten

Das Webinterface soll den Browser nicht wesentlich belasten. Auch lokal ausgeführter Javascript-Code sollte auf einem zeitgemässen Client-Computer keinen nennenswerten Teil der Rechenleistung beanspruchen.

### 5.3.6. Änderbarkeit

#### Analysierbarkeit

Die Applikation soll auch nach Projektabschluss betreubar sein. Daher sollen folgende Anforderungen erfüllt werden:

- Die Coding-Standards sollen eingehalten werden (PEP8, siehe auch Projektplan)
- Klassen und nicht-triviale Methoden sollen im Code dokumentiert werden (Doc-Strings)
- Das Architekturdokument soll gegen Ende des Projektes dem effektiv implementierten Programm entsprechen. Es soll nicht nur als Grundlage zur Entwicklung dienen, sondern auch als Nachschlagewerk für spätere Entwickler.

#### Modifizierbarkeit

Es soll bei der Entwicklung darauf geachtet werden, zukünftige Erweiterungen nicht durch Design-Entscheide auszuschliessen. Einen Teil davon wird durch die geplante JSON-API bereits erfüllt.

### 5.3.7. Testbarkeit

Der geschriebene Code wird mittels Unit-Tests getestet. Falls zu einem späteren Zeitpunkt Features hinzugefügt oder verändert werden, kann mithilfe dieser Tests sichergestellt werden, dass die restlichen Funktionen weiterhin korrekt ausgeführt werden.

### 5.3.8. Übertragbarkeit

#### Anpassbarkeit

Die Applikation soll einfach an persönliche Bedürfnisse angepasst werden können. Die meisten heute gängigen Browser unterstützen Custom-Stylesheets, die viele Änderungen am Look-and-feel ermöglichen. Die Applikation soll internationalisiert entwickelt werden, Bezeichnungen und Fehlermeldungen sollen in separate Dateien ausgelagert werden, die eine spätere Erweiterung um zusätzliche Sprachen möglich machen.

#### Installierbarkeit

Die benötigte Software für die Applikation sind die folgenden Programme:

- Apache Webserver
- Python 2.7
- Django 1.5
- Ein Webbrowser

Alle diese Programme können unter Linux mithilfe eines Paket-Managers (apt, yum) auf einfachste Art installiert werden.

#### Konformität

Der Python-Code richtet sich nach den Vorgaben des PEP8-Dokuments (siehe Auch: Projektplan). Der restliche Code soll sich möglichst an die gängigen Standards im Webbereich halten, insbesondere valides HTML-Markup in generierten Inhalten.

## 5.4. Anforderungen an die Plattform

### 5.4.1. Open-Source

Das geplante Tool muss zwingend auf Open-Source Software aufbauen, als Webserver muss Apache unter Linux verwendet werden.

### 5.4.2. Standardisierung

Das Tool soll sich an etablierte Standards halten, damit es mit allen gängigen Browsern (vor allem natürlich Open-Source Browser wie Firefox oder Chromium) betrachtet und bedient werden kann.

### 5.4.3. Einfaches Setup

Die Einrichtung des fertigen Tools soll möglichst simpel gehalten werden, oder gar automatisierbar sein. Dies gilt auch für die zugrundeliegende Plattform.

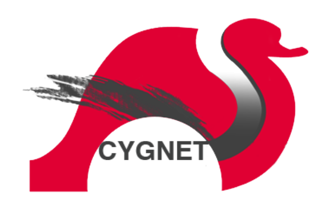






# Cygnet

# Architektur




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser:** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 6. Einführung

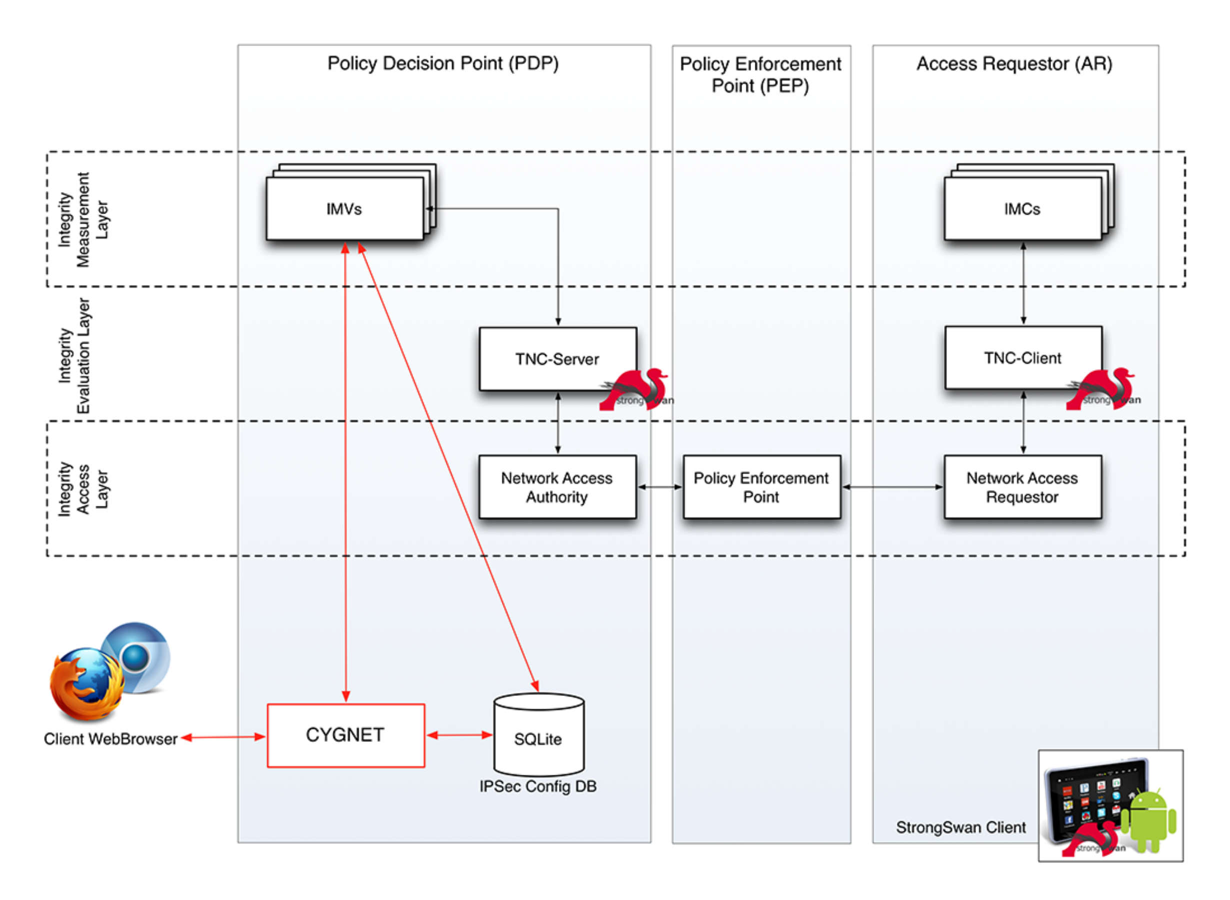



Abbildung 6.1.: Übersicht der Komponenten

Das Malware Database Management Tool Cygnet soll eng mit der bestehenden strongSwan-Software zusammenarbeiten. Es erweitert die bestehende Konfigurationsdatenbank um ein web-basiertes Management Interface, das die bisherige Verwaltung via Kommandozeilentool ablösen soll. Zudem wird ein Regelwerk implementiert, welches dem Administrator erlaubt, Richtlinien für strongSwan Clients festzulegen, die vom TNC-Server durchgesetzt werden, bevor der Client ins Netzwerk eingelassen wird.

## 6.1. Komponenten

### 6.1.1. Policy Decision Point (PDP):

In der Abbildung 1.1 zeigt der linke eingerahmte Bereich den Policy Decision Point. Die Applikation Cygnet befindet sich somit auf einem Server, der die Aufgabe hat die Messwerte

eines Access Requestors zu sammeln und mithilfe von Policies eine Zugriffsentscheidung zu formulieren. Diese Entscheidung wird anschliessend der ausführenden Stelle für den Zugriff mitgeteilt. Auf dem PDP existieren mehrere sogenannte Integrity Measurement Verifier (IMV) für die unterschiedlichen Sicherheitskomponenten. Sie vergleichen die übermittelten Messwerte anhand der in den Policies festgelegten Regeln und teilen ihr Ergebnis dem TNC-Server im PDP mit. Dieser trifft mit den Teilergebnissen eine Gesamtentscheidung über die Integrität des Rechnersystems und teilt diese Entscheidung dem Policy Enforcement Point mit.

### 6.1.2. Policy Enforcement Point (PEP):

Stellt das TNC-Element am Eintrittspunkt des Netzwerkes dar. Seine Aufgaben sind die Entgegennahme und Weiterleitung von Verbindungsanfragen sowie die Ausführung der Handlungsentscheidung des PDPs.

### 6.1.3. Access Requestor (AR):

Das Rechnersystem, über das eine Netzwerkverbindung zu einem TNC-Netzwerk aufgebaut werden soll, wird Access Requestor genannt. Auf dem Access Requestor befinden sich TNC-Komponenten für Verbindungsanfrage, Messwertübermittlung und Messung. In dieser Arbeit beschränken wir den AR auf ein Android-Gerät oder einen Rechner mit einer installierten Ubuntu-Distribution.

# 7. Ziele & Prinzipien

## 7.1. Bestehendes System

Zu Beginn der Bachelorarbeit existierte bereits eine Applikation die einen Teil der Funktionalität bietet. Sie bestand im Wesentlichen aus einer SQLite Datenbank sowie einem Kommandozeilentool, mit welchem die Daten manipuliert werden konnten. Diese Applikation dient als Grundlage für das geplante Cygnet-Tool. Bestehende Features sollen allesamt übernommen oder erweitert werden.

### 7.1.1. Schnittstelle Cygnet ↔ IMV

Die Datenbank wird auch als Datenablage für die bestehenden strongSwan IMVs verwendet, die ihre Resultate und Referenzwerte darin abspeichern. Für diese Zugriffe soll eine Schnittstelle definiert werden, damit sich die Datenzugriffe von Cygnet und den IMVs nicht gegenseitig stören.

### 7.1.2. Stabilität gegenüber IMVs

Cygnet ist als Administrations- und Unterstützungstool für strongSwan und dessen TNC-Server gedacht. Daher steht die Funktion des TNC im Vordergrund. Eine Fehlfunktion von Cygnet soll daher nicht den Betrieb des strongSwan Gateways beeinträchtigen.

## 7.2. Flexibilität für Administratoren

Netzwerke gibt es in beinahe beliebiger Vielfalt und mit enorm unterschiedlichen Anforderungen, daher ist auch deren Administration eine entsprechende Herausforderung. Cygnet soll dem strongSwan Administrator soweit wie möglich zur Hand gehen und ihn bei der Konfiguration unterstützen, ohne aber seine Möglichkeiten einzuschränken. Cygnet soll daher bewusst so implementiert werden, dass für den Administrator jeweils mehr als eine Möglichkeit existiert, eine Richtlinie umzusetzen. Die grafische Oberfläche soll jedoch übersichtlich und soweit wie möglich intuitiv bedienbar bleiben.

# 8. Datenbanken

Das Datenmodell wurde aus dem bestehenden Modell erarbeitet; Namensgebung und Struktur wurden nach Möglichkeit beibehalten. Die einzelnen Ausschnitte des Diagramms werden weiter unten genauer beschrieben.

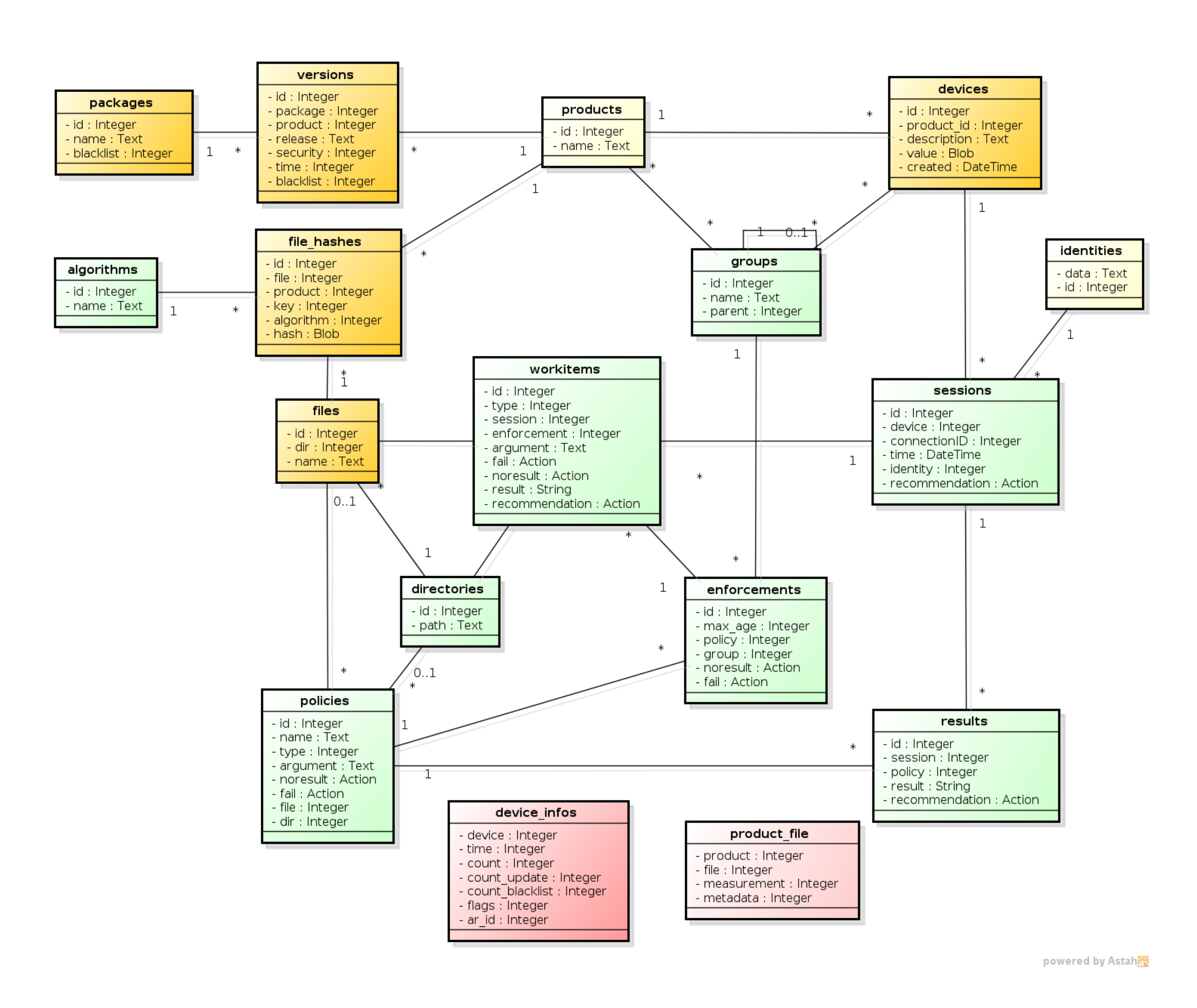



Abbildung 8.1.: ER-Diagramm

| **Legende** | |
|---|---|
| Grün: | Neue Tabelle |
| Hellgelb: | Bestehende Tabelle |
| Dunkelgelb: | Bestehende Tabelle, veränderte Feld-Definitionen |
| Rot: | Bestehende, neu nicht mehr benötigte Tabelle |

## 8.1. Dateihashes

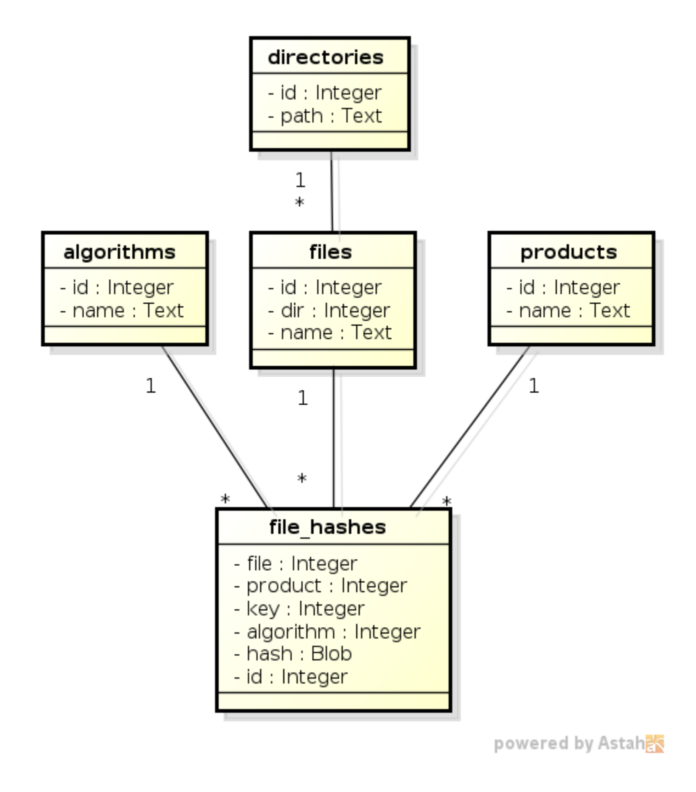



Abbildung 8.2.: ER-Diagramm: Dateien & Hashes

Die in 8.1 gezeigten Tabellen enthalten die Referenzwerte für Dateihashes die mit den effektiven Werten auf den strongSwan-Clients verglichen werden. Die Dateipfade werden in den zwei Tabellen `directories` und `files` abgelegt. `files.name` ist jeweils nur der `basename` + Suffix der Datei, während `directories.path` den Ordnerpfad ohne trailing slash enthält.
Die Einträge in `file_hashes` werden über ihre Beziehung zu `products` einer bestimmten Betriebssystemversion zugeordnet.

### 8.1.1. Algorithmen

Die `algorithms`-Tabelle enthält genau die folgenden Einträge, die Primärschlüssel wurden aus Kompatibilitätsgründen mit strongSwan so gewählt. Ihr Zweck ist im Wesentlichen dem Administrator im Web-Frontend einen leserlichen Algorithmus-Namen zu präsentieren. Eine allfällige Erweiterung der Tabelle (z.B. bei der Verabschiedung der SHA-3 Hashfamilie) kann so einfach durchgeführt werden.

| ID | Name |
|---|---|
| 65536 | SHA1_IMA |
| 32768 | SHA1 |
| 16384 | SHA256 |
| 8192 | SHA384 |
| 0 | NONE |

## 8.2. Packages

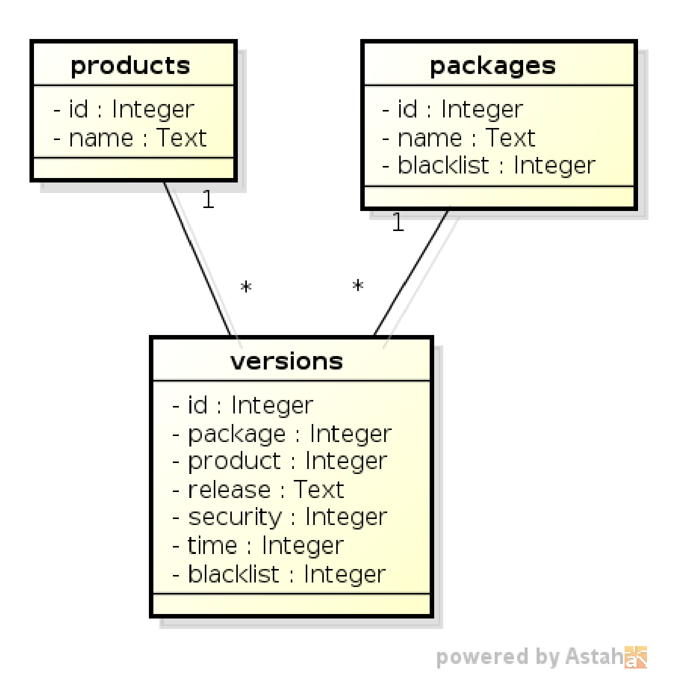



Abbildung 8.3.: ER-Diagramm: Packages

Betriebssystem-Softwarepakete werden in den Tabellen `packages` und `versions` gespeichert. Die Spalte `packages.name` enthält den Paketnamen (z.B. `mysql-common`), während in der Versionstabelle jeweils die aktuellste Versionsnummer pro Betriebssystem gespeichert wird. Es können mehrere Versionseinträge pro Produkt und Paket existieren; je einer für das aktuellste optionale Update und das aktuellste sicherheitsrelevante Update.
Ob ein Paket `blacklist`ed ist, ist ebenfalls hier gespeichert und kann somit unterschiedlich für jedes Betriebssystem und jede Programmversion festgelegt werden.

## 8.3. Gruppen & Policies

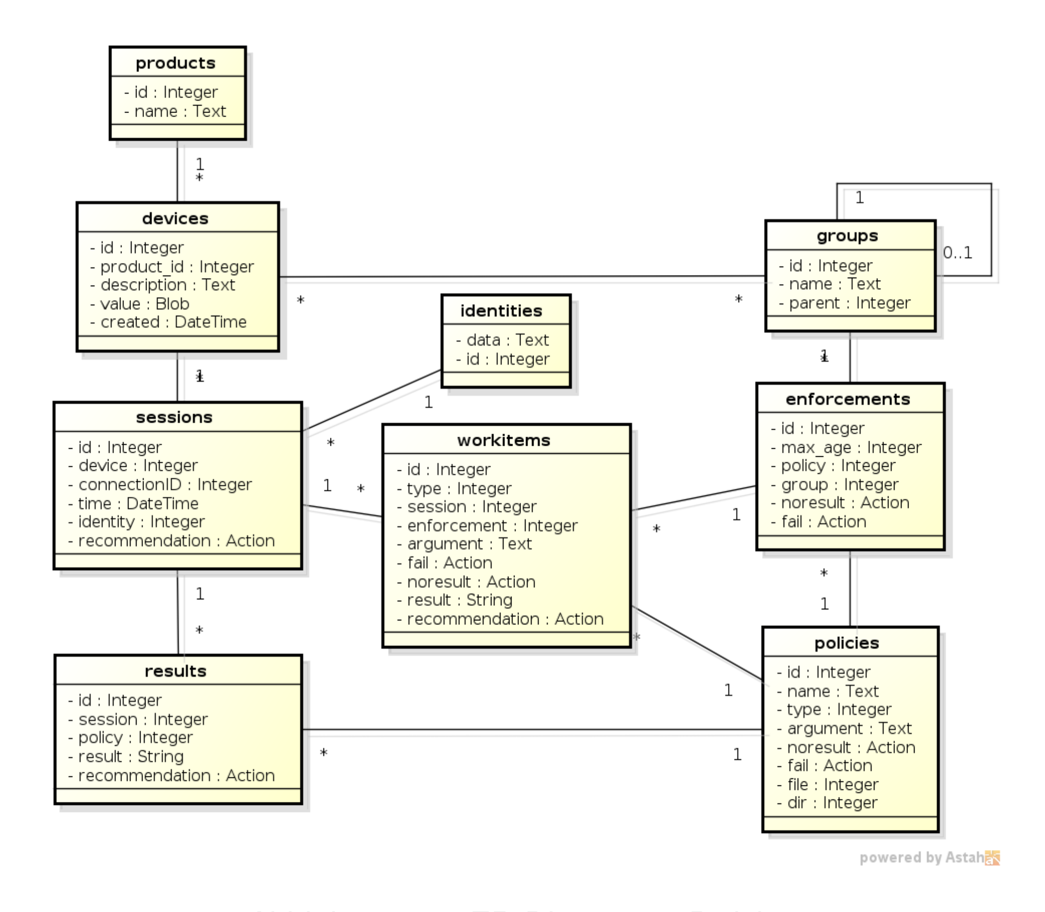



Abbildung 8.4.: ER-Diagramm: Policies

Das Kernstück von Cygnet. Die genauen Details sind in einem separaten Kapitel 9: Policy-Management auf Seite 30 beschrieben.

## 8.4. Django Metadaten

Das Django-Framework benötigt einige eigene Datenbanktabellen, um interne Metadaten zu speichern. Da mit Cygnet und den strongSwan IMVs bereits zwei Aktoren auf derselben Datenbasis arbeiten, wurde ein Datenbankrouter in Python realisiert, der alle Django-Metadaten in eine separate SQLite Datenbank auslagert und Zugriffe darauf entsprechend umlenkt.

# 9. Policy-Management

Als Policy wird eine einzelne Regel bezeichnet, deren Bedingung ein Client zu erfüllen hat, um ins interne Netzwerk vorgelassen zu werden. Die folgenden Typen von Policies sind zur Zeit definiert:

| ID | Name | Parameter (String) |
|---|---|---|
| 0 | File Hash | File ID |
| 1 | Dir Hash | Directory ID |
| 2 | Listening Port TCP | Port Range |
| 3 | Listening Port UDP | Port Range |
| 4 | File Exist | File ID |
| 5 | Not File Exist | File ID |
| 6 | Missing Update | - |
| 7 | Missing Security Update | - |
| 8 | Blacklisted Package | - |
| 9 | OS Settings | - |
| 10 | Deny | - |

Tabelle 9.1.: Policy-Typen

- FileHash: Prüft, ob ein File verändert wurde, indem es seinen Hash mit einem Referenzwert aus der Datenbank vergleicht.
- DirHash: Vergleicht alle FileHashes aller bekannten Dateien im gegebenen Verzeichnis mit den entsprechenden Referenzwerten.
- ListeningPort (TCP/UDP): Prüft, ob auf der angegebenen Port-Range Listening Sockets registriert wurden. Der Range Parameter hält sich dabei an das Format `\d{1,5}(-\d{1,5})?(,(\d{1,5})(-\d{1,5})?)*`.
- FileExist: Prüft, ob die angegebene Datei existiert.
- NotFileExist: Prüft, ob die angegebene Datei NICHT existiert.
- MissingUpdate: Prüft, ob alle installierten Pakete auf der aktuellsten Version sind.
- MissingSecurityUpdate: Prüft, ob alle installierten Pakete die letzten Security-Updates erhalten haben.
- BlacklistedPackage: Prüft, ob ein unerlaubtes Paket installiert ist.
- OSSettings: Kontrolliert bestimmte Betriebssystemeinstellungen.
- Deny: Schlägt per Definition immer fehl und dient zum Ausschliessen eines Geräts.

## 9.1. Konfigurieren der Policies

### 9.1.1. Definition von Gruppen

Der Administrator kann seine Client-Geräte in Gruppen organisieren, die eine Hierarchie unterstützen. Eine Gruppe kann eine Eltern-Gruppe haben. Richtlinien der Eltern-Gruppe werden auf die Kind-Gruppe vererbt. Eine Einteilung nach Organisationsstruktur ("Dozenten, Studenten") oder nach technischen Eigenschaften ("Smartphones", "Tablets") oder eine Kombination von Beidem ist so realisierbar.
Nach Installation von Cygnet existiert bereits eine Default-Gruppe, die nur umbenannt oder in der Hierarchie eingeordnet, aber nicht gelöscht werden kann. Sie dient als Standard-Gruppe für die Default-Policies (siehe 9.1.5).

### 9.1.2. Policies

In einem zweiten Schritt können Policies definiert werden. Eine Policy kann einem Typen aus Tabelle 9 entsprechen. Weiter können eine Fail-Action und eine Noresult-Action angegeben werden. Dort stehen vier Optionen zur Auswahl:

| Option | Bedeutung |
|---|---|
| ALLOW | Es wird empfohlen, den Client ins Netz zuzulassen. |
| ISOLATE | Es wird empfohlen, den Client in einem abgesondertes Netz zu isolieren. |
| BlOCK | Es wird empfohlen, dem Client jeglichen Zugriff zu verweigern. |
| NONE | Keine Empfehlung. |

Tabelle 9.2.: Actions

Die Fail-Action wird empfohlen, wenn der Client die Anforderung der Policy nicht erfüllt. Die Noresult-Action wird angewendet, wenn die Policy nicht auf dem Client ausgeführt werden kann. Wenn mehrere Policies auf einen Client wirken und unterschiedliche Empfehlungen abgeben, wird die konservativste Empfehlung durchgesetzt. (BLOCK vor ISOLATE vor ALLOW).
"NONE" als Empfehlung macht vor allem in Kombination mit anderen Policies Sinn. Wenn keine Policy eine Empfehlung abgibt, hängt die endgültige Entscheidung von der StrongSwan-Implementation ab.

### 9.1.3. Enforcements

Als dritter und letzter Schritt werden die Policies und Gruppen mit Enforcements verlinkt. Ein Enforcement erzwingt eine Policy auf einer Gruppe. Zusätzlich kann angegeben werden, wie oft eine Policy geprüft werden soll, als Anzahl Tage. War eine Messung bei einem Client erfolgreich, wird die Policy erst wieder geprüft, wenn die letzte Messung weiter als die angegebene Anzahl Tage zurückliegt. So kann der Login-Prozess für Clients, die sich an die Regeln halten, verkürzt werden. Wenn das Ergebnis der letzten Messung nicht erfolgreich war (oder kein Resultat vorliegt) wird die Messung auf jeden Fall durchgeführt.

Für jedes Enforcement kann die Fail- oder Noresult-Action überschrieben werden, um je nach Gruppe unterschiedlich hart zu reagieren. Per default wird die Einstellung von der gewählten Policy geerbt.

### 9.1.4. Sessions, Workitems und Resultate

Wenn sich ein Client beim System anmelden will, wird von StrongSwan eine neue Session für das Gerät erzeugt. Cygnet generiert zu diesem Zeitpunkt eine Liste von Workitems, die eine konkrete Auftragsliste für die IMVs darstellt. Die Workitems sind eine Aggregation aus allen Policies die auf den Client wirken und getestet werden müssen. Die IMVs führen die Messungen durch und schreiben die Resultate zurück in die Datenbank. Cygnet weist die Resultate der Session zu.

### 9.1.5. Produkte und Default-Policies

Es wird für jedes Client-Gerät gespeichert, welches Produkt (lies: “Betriebssystem”) installiert ist. Diese Produkte können in Cygnet verwendet werden um eine Richtlinie für Geräte zu definieren, die sich das erste Mal am System anmelden.
Es können pro Produkt eine oder mehrere Standard-Gruppen definiert werden. Wenn sich ein neues Gerät anmeldet, wird es automatisch aufgrund seines Produkts in die zugehörigen Standard-Gruppen eingeteilt und die darauf wirkenden Enforcements werden auf das Gerät angewandt.
Wenn ein Gerät mit einem bisher unbekannten Produkt sich anzumelden versucht, wird das Produkt bei Cygnet registriert und das Gerät in die immer existierende Standard-Gruppe (siehe 21.2) eingeteilt.

### 9.1.6. Softwarepakete, Versionen und Blacklisting

Die Liste von Software-Paketen wird automatisch anhand der bei Clients installierten Software aktualisiert. Software-Pakete sind immer einem Produkt (lies: “Betriebssystem”) zugeordnet und können sowohl pro Version, als auch global auf eine Blacklist gesetzt werden, die als Grundlage für den Policy-Typ “Blacklisted Package” dient.

# 10. Externes Design

## 10.1. Paper Prototypes

Die folgenden “Screenshots” geben einen Überblick, wie das Webinterface von Cygnet später aussehen könnte:

### 10.1.1. Login

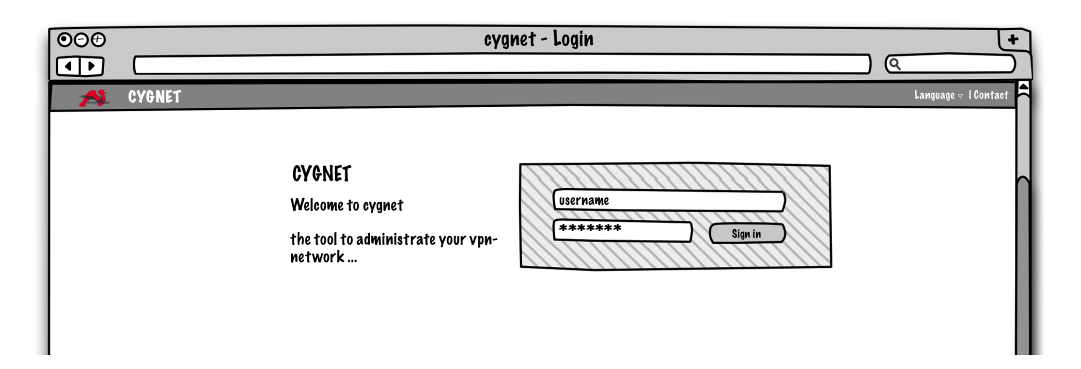



Abbildung 10.1.: Anmeldeseite

Startseite der Cygnet-Anwendung. Der Benutzer gibt hier seine Login-Daten ein.

### 10.1.2. Overview

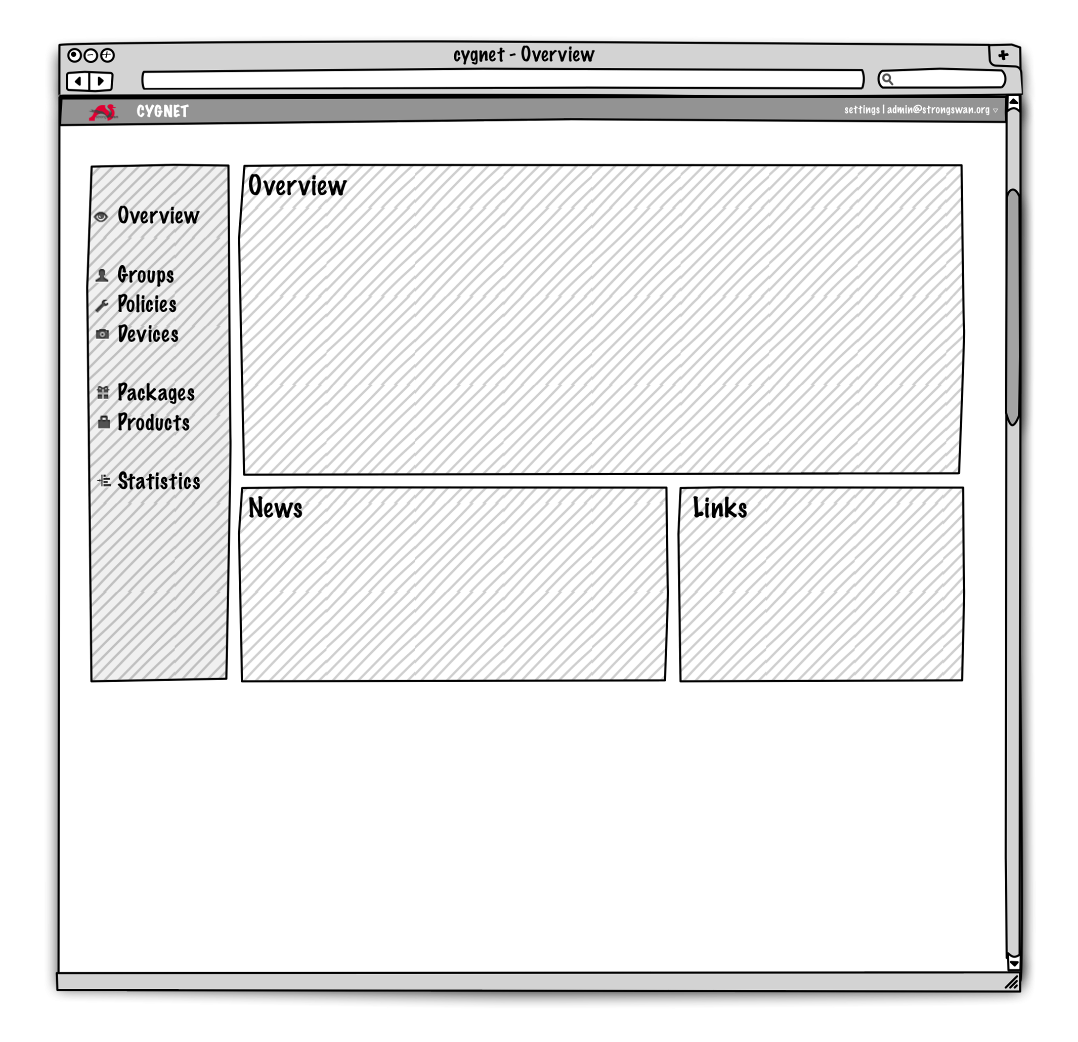



Abbildung 10.2.: Übersicht

Dies ist die Hauptseite, in welcher auf der linken Seite die Navigation zu allen wichtigen Funktionen führt. Hier erhält der Administrator diverse wichtige Informationen über das laufende System sowie Links zu informativen Webseiten.

### 10.1.3. Gruppenverwaltung

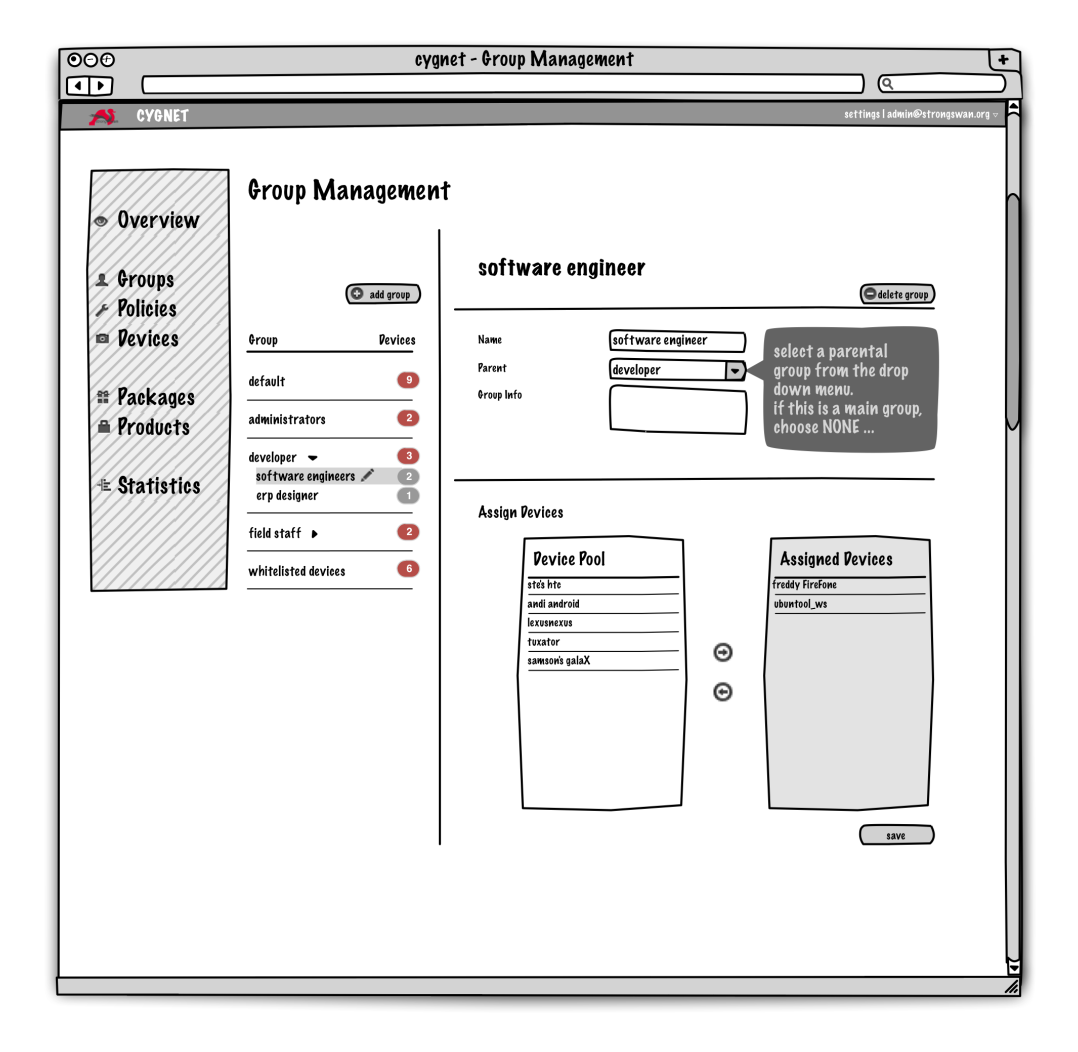



Abbildung 10.3.: Gruppenansicht

Die Gruppenverwaltung bietet dem Administrator die Möglichkeit die Informationen einer Gruppe zu bearbeiten. Nach Auswahl einer Gruppe kann er neben Name und Beschreibung auch die Zugehörigkeit der Elterngruppe ändern. Im unteren Bereich können die Geräte dieser Gruppe hinzugefügt oder entfernt werden.

### 10.1.4. Policy-Management

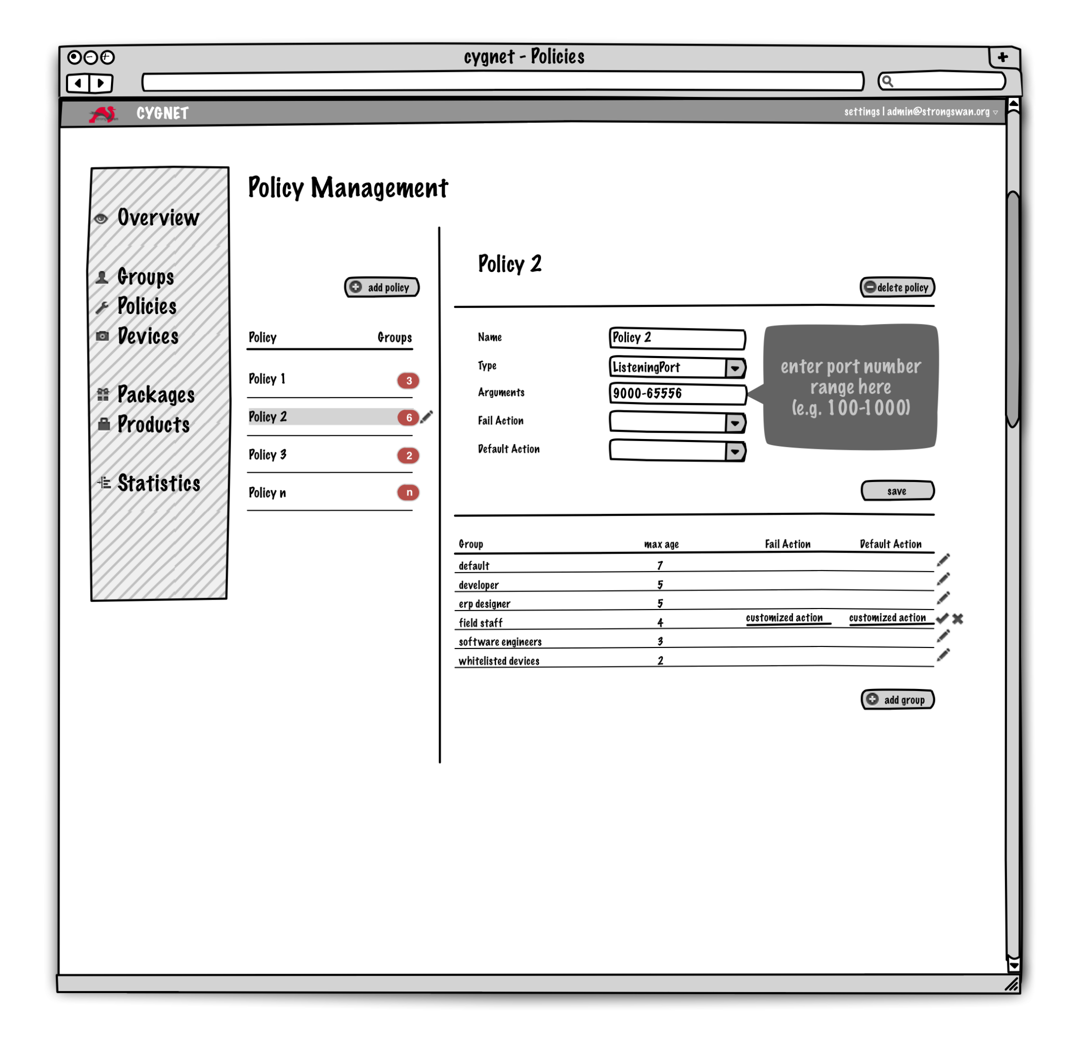



Abbildung 10.4.: Policyansicht

In diesem Bereich verwaltet der Administrator seine Policies. Bei den allgemeinen Informationen (Name, Typ, etc.) erhält er durch Tooltips, welche bei einem Mouse-Over erscheinen, eine Eingabehilfe über das entsprechende Feld. In der unteren Tabelle stehen dem Administrator Einstellungsmöglichkeiten für der Policy zugewiesenen Gruppen zur Verfügung.

### 10.1.5. Geräteverwaltung

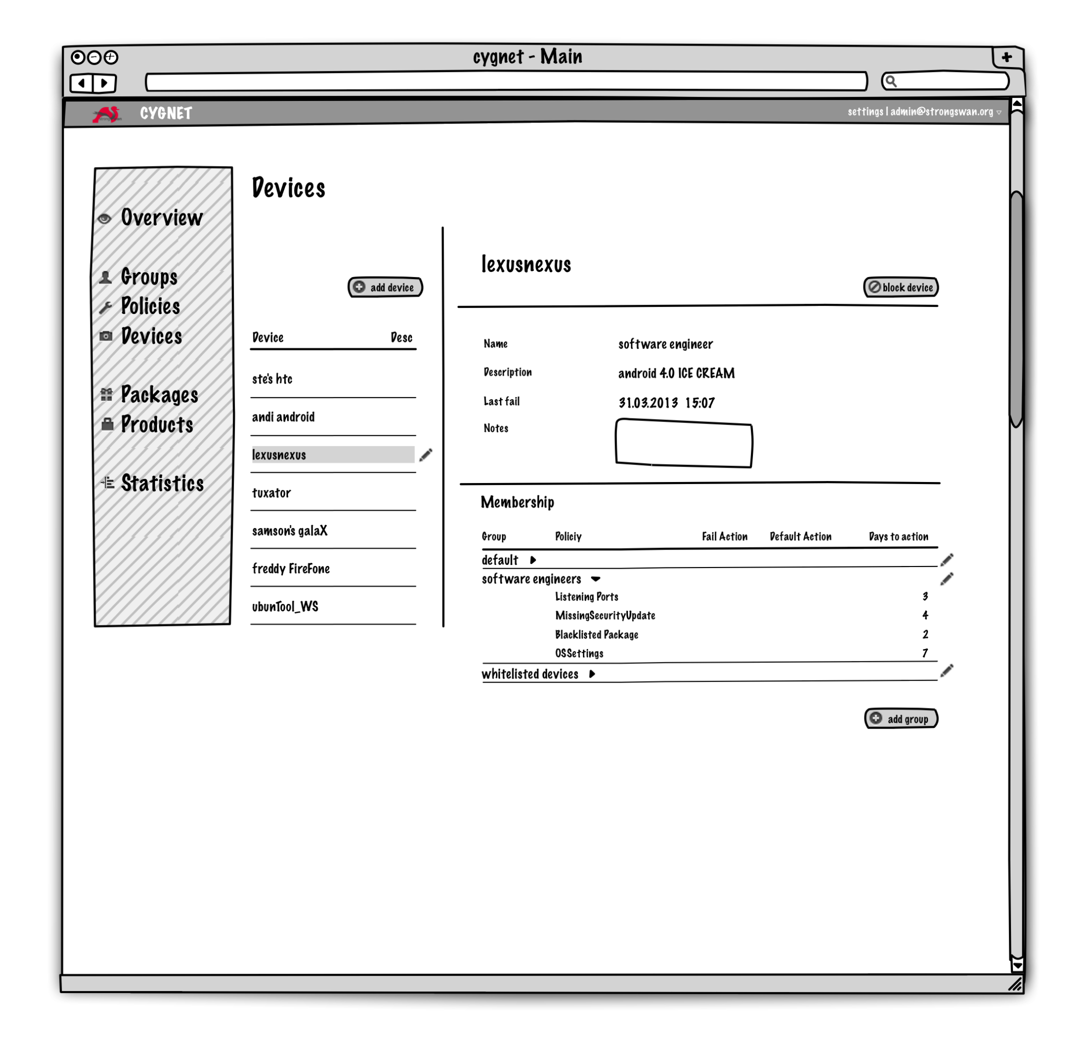



Abbildung 10.5.: Geräteansicht

Um Geräte den Gruppen und somit den Policies zuzuweisen, kann auf der Geräteansicht ein Gerätename ausgewählt werden. Hier sind für den Administrator auch Informationen über das Gerät ersichtlich.

## 10.2. Generische Views

Django ist nach dem Model-Template-View (MTV) Pattern aufgebaut. MTV orientiert sich am bekannten Model-View-Controller Pattern (MVC). Somit wird die Möglichkeit geboten, Templates für die verschiedenen Seiten anzulegen. Diese Django Templates sind einfache Python Objekte, deren Konstruktor einen String erwartet. Mit Hilfe eines Context Objekts werden dann die Platzhalter im Template beim Rendern der HTML-Seite durch die gewünschten Werte ersetzt.
Ein Master-Template übernimmt die Grundgestaltung des GUIs und wird den einzelnen Templates vererbt.

## 10.3. GUI-Gestaltung durch das Bootstrap-Framework

Mit dem Bootstrap-Framework wird die Oberflächengestaltung von GUI-Elementen wie Grid-Systeme, Navigationsbereiche, Tabellenansichten und Formulare auf HTML und CSS basierend vereinheitlicht und somit vereinfacht. Bootstrap weist eine Kompatibilität für alle wichtigen Browser auf und bietet mit diesem sogenannten Responsive Webdesign auch eine dynamische Anpassung der Oberflächenelemente auf Smartphones und Tablets an.

# 11. Schnittstellen

## 11.1. Ablauf TNC-Prozess

Der Verbindungsaufbau wird jeweils von einem Android-Gerät ausgelöst, das sich mit dem VPN-Netzwerk verbinden will. Der Ablauf wird im folgenden Diagramm dargestellt:

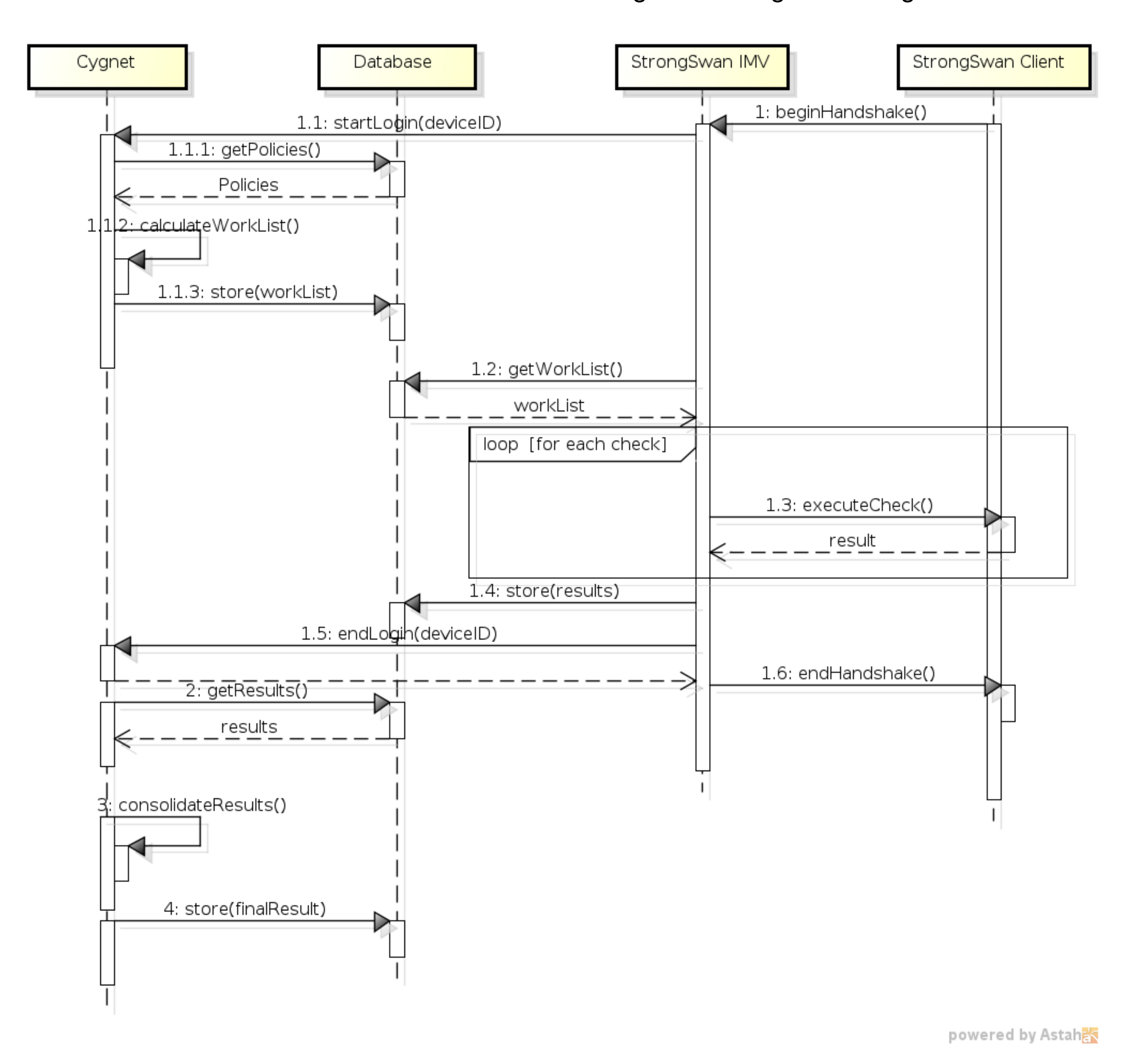



Abbildung 11.1.: Sequenzdiagramm TNC-Prüfung

1. Der Verbindungsaufbau wird durch den Android-Client ausgelöst
   1.1) Der (Haupt-)IMV gibt den Login-Vorgang bei Cygnet bekannt
      1.1.1. Cygnet liest anhand der Gruppenzugehörigkeiten des Geräts dessen geltenden Regeln/Policies aus
      1.1.2. Aus den Policies wird dann die Liste von WorkItems errechnet, die auf dem Client ausgeführt werden muss
      1.1.3. Die WorkList wird für den IMV in einer separaten Tabelle in der Datenbank abgelegt
   1.2) Der IMV holt sich die WorkList aus der Datenbank
   1.3) Die einzelnen Checks werden auf dem Client ausgeführt und die Resultate an den IMV zurückgeliefert
   1.4) Der IMV speichert die Resultate zu den einzelnen WorkItems in der Datenbank
   1.5) Anschliessend gibt der IMV die Kontrolle der Datenbank zurück an Cygnet
   1.6) Der IMV signalisiert dem Client das Ende des Handshakes und baut je nach Resultat die VPN-Verbindung auf oder weist den Client ab
2. Cygnet liest die mit Resultaten gefüllte WorkList aus
3. Die Resultate werden ausgewertet. . .
4. . . . und in der Datenbank abgelegt

## 11.2. Datenmodell

SQLite kennt weder formale Datentypen, noch wird referentielle Integrität erzwungen. Der OR-Mapper von Django forciert die im Modell definierten Anforderungen an die Daten, und da nebst Cygnet auch die IMVs auf die Datenbank zugreifen, ist es wichtig, dass auch diese sich an die formale Definition der Daten halten. Die komplette Definition (in gängiger SQL-Notation) ist daher im Anhang (Seite 93) angehängt.

## 11.3. Views und URLs

In diesem Abschnitt sind die URLs aufgeführt, die in Cygnet aufrufbar sind. Eine URL beschreibt bei Django per Definition nicht eine Datei, sondern einen Python-Funktionsaufruf (eine View) der eine HttpResponse kreiert und zurückgibt.
Jede Benutzerview ist im Code in mehrere Phasen gegliedert:

1. HTTP-Operation prüfen (implizit via Decorator)
2. Parameter parsen & prüfen
3. Datenbank-Operation ausführen
4. Antwort-Kontext aufbauen
5. Antwort an Client senden

## 11.4. User - Views

Dies sind die Views, welche dem Benutzer präsentiert werden. Sie machen das Frontend für den Benutzer aus. Auf den folgenden Seiten werden exemplarisch die Views für die Gruppenverwaltung dokumentiert.

| URL | Beschreibung |
|---|---|
| `/groups` | Gruppenübersicht |
| `/groups/<ID>` | Detailansicht Gruppe |
| `/groups/add` | Neue Gruppe |
| `/groups/save` | Gruppe speichern/aktualisieren |
| `/groups/<ID>/delete` | Gruppe löschen |

Eine passende Struktur von Views existiert jeweils für `/devices`, `/policies`, `/enforcements`, `/packages`, `/products` und `/files`.

### 11.4.1. Gruppenübersicht

- **URL:** `/groups`
- **Verb:** GET
- **Parameter:** Keine
- **Returns:**
  - Response-Code:
    - HTTP 200: OK
  - Response-Format: HTML

### 11.4.2. Gruppendetail

- **URL:** /groups/<group>
- **Verb:** GET
- **Parameter:**

| Methode | Name | Beschreibung | Format |
|---|---|---|---|
| REST | group | ID der Gruppe | ^\d+$ |

- **Returns:**
  - Response-Code:
    - HTTP 200: OK
    - HTTP 404: No such group
  - Response-Format: HTML

### 11.4.3. Neue Gruppe

- **URL:** /groups/add
- **Verb:** GET
- **Parameter:** Keine
- **Returns:**
  - Response-Code:
    - HTTP 200: OK
  - Response-Format: HTML

### 11.4.4. Gruppe speichern

- **URL:** /groups/save
- **Verb:** POST
- **Parameter:**

| Methode | Name | Beschreibung | Format |
|---|---|---|---|
| POST | groupId | ID der Gruppe, optional | ^\d+$ |
| POST | parent | ID der Parent-Gruppe, optional | ^\d+$ |
| POST | name | Name der Gruppe | ^[\S ]{1,50}$ |
| POST | memberlist | IDs von Member-Devices | ^\d+((,\d+,)*,\d+)?$ |

- **Returns:**
  - Response-Code:
    - HTTP 302: Found (Redirect zu /groups/<groupID>)
  - Response-Format: N/A

### 11.4.5. Gruppe löschen

- **URL:** /groups/<group>/delete
- **Verb:** POST
- **Parameter:**

| Methode | Name | Beschreibung | Format |
|---|---|---|---|
| POST | groupId | ID der Gruppe | ^\d+$ |

- **Returns:**
  - Response-Code:
    * HTTP 302: Found (Redirect zu /groups)
  - Response-Format: N/A

## 11.5. API - Views

Die API-Views dienen als Schnittstelle zwischen dem Management-IMV und Cygnet. Ein Client sollte sich an folgende Definitionen halten:

### 11.5.1. Start-Session

- **URL:** /cmd/start_session
- **Verb:** HEAD (,GET)
- **Parameter:**

| Methode | Name | Beschreibung | Format |
|---|---|---|---|
| GET | connectionID | TNC Connection ID | ^\d+$ |

- **Returns:**
  - Response-Code:
    * HTTP 200: OK (Workitems created)
    * HTTP 400: Invalid argument
  - Response-Format: None
  - Response-Body: None

### 11.5.2. End-Session

- **URL:** `/cmd/end_session`
- **Verb:** HEAD (,GET)
- **Parameter:**

| **Methode** | **Name** | **Beschreibung** | **Format** |
|---|---|---|---|
| GET | connectionID | TNC Connection ID | `^\d+$` |

- **Returns:**
  - Response-Code:
    - HTTP 200: OK (Session finished)
    - HTTP 400: Invalid argument
    - HTTP 404: No such session
  - Response-Format: None
  - Response-Body: None

# 12. Input-Validierung

Die Validierung von User-Input ist ein wichtiges Thema für Internetapplikationen und kann, wenn nicht richtig implementiert, eine kritische Schwachstelle sein, die potentielle Angreifer ausnutzen können.
Clientseitig werden alle Formulare mithilfe von Javascript-Code auf ihre (syntaktische) Richtigkeit geprüft, bei einem Fehler wird der Benutzer darauf hingewiesen und hat die Möglichkeit, den Fehler zu korrigieren. Erst nachdem alle Fehler beseitig sind, kann der Benutzer das Formular an den Server schicken. Doppelte Eingaben in Felder, die als unique definiert sind, werden via AJAX-Request getestet, bevor der Benutzer das Formular absenden kann.
Die clientseitige Prüfung des Inputs reicht nicht aus; ein Angreifer kann jederzeit beliebige HTTP-Request kreieren, die keiner Javascript-Prüfung unterliegen. Deshalb wird der Input auf der Serverseite ein zweites Mal überprüft. Tritt bei dieser Überprüfung ein Fehler auf, reagiert der Server mit einem HTTP-Statuscode 400 (Bad Request). Es gibt keine spezifische Fehlermeldung für den Benutzer. Das GUI muss daher gewährleisten, dass keinerlei “schlechte” Inputs an den Server gelangen.

# 13. Deployment

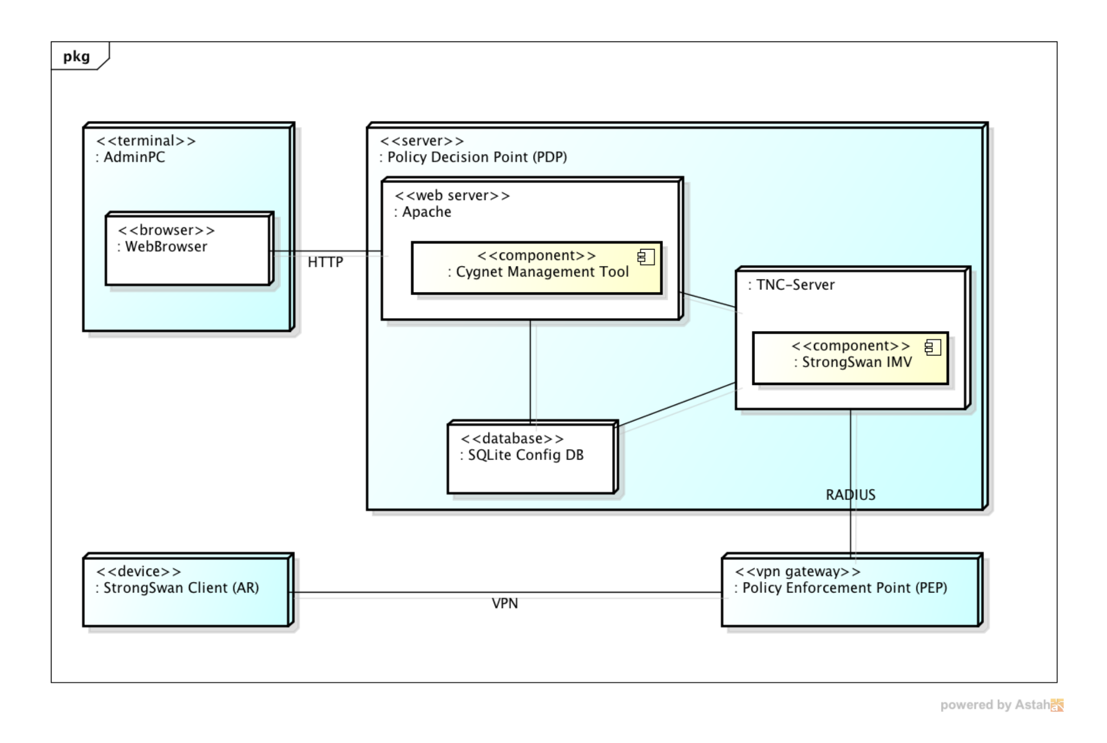



Abbildung 13.1.: Deployment Diagramm

In diesem Diagramm ist die physische Verteilung von Komponenten auf Verarbeitungsknoten und die physische Netzwerkkonfiguration zwischen den Knoten ersichtlich.
Die Komponente “Cygnet Management Tool”, dabei handelt es sich um einen “Execution Environment Note” (EEN), ist eine Software-Ressource, die auf dem PDP-Server läuft und die Funktionalität des Management Tools zur Verfügung stellt. Eine physische Verbindung wird zum Webbrowser des Administrators über HTTP aufgebaut.
Auf dem Policy Decision Point (PDP), welcher als “Device Node” ein Geräteknoten darstellt, sind weitere Ausführungsknoten untergebracht, wie die SQLite-Datenbank und der TNC-Server, die beide mit der Cygnet Software kommunizieren.
Wie in der Einleitung erwähnt, steht der TNC-Server mit dem strongSwan Client (AR) über den Policy Enforcement Point (PEP) in Verbindung.

Eine konkrete Anleitung, wie Cygnet zu installieren ist, findet sich im Part Deployment. (Seite 61)

# 14. Entscheidungen

## 14.1. Verzicht auf Domänenmodell

Es wurde für das Projekt bewusst auf ein Domänenmodell verzichtet. Ein Domänenmodell abstrahiert in der Regel die Problemdomäne, sprich die Realität. Da aber die Problemdomäne in diesem Fall bereits eine Abstraktion auf hohem Niveau ist, wurde kein weiteres "Level of indirection" aufgesetzt. Als gute Diskussionsgrundlage im Gespräch mit unserem Betreuer diente das Datenmodell der bestehenden Datenbank. Es wurde im Projektverlauft stetig erweitert und verbessert.

## 14.2. Redundanzen in Workitems-Tabelle

Die Schnittstelle zwischen Cygnet und StrongSwan stellte sich als Herausforderung dar, deshalb wurden zur Vereinfachung die Workitems-Tabelle definiert. Sie stellt eine Konsolidierung der restlichen Cygnet-Tabellen dar und bieten den IMVs eine Liste von Aufträgen, die abzuarbeiten sind. In der Tabelle sind bewusst Redundanzen eingebaut, die Datennormalisierungsregeln verletzen. Die Verletzungen wurden in Kauf genommen, da die Tabelle nur beim Login eines Clients einige Einträge generiert und diese bei erfolgreicher oder auch missglückter Verbindung anschliessend wieder gelöscht werden. Somit ist diese Tabelle für den Grossteil der Zeit leer.

## 14.3. BLOCK vor ISOLATE

Je nach Auslegung ist die Massnahme ISOLATE für einen Client "härter" als BLOCK. Nach Auslegung des Projektteams ist aber BLOCK schwerwiegender, da der Client bei ISOLATE zumindest einen teilweisen Zugriff auf das Netzwerk bekommt. Es wurde daher bewusst darauf verzichtet, die Reihenfolge und/oder Priorität dieser Recommendations konfigurierbar zu machen.

## 14.4. Unlöschbare Gruppe #1

Es wurde entschieden, dass wenn ein Client kein (erkennbares) Betriebssystem installiert hat oder keine Product-Default-Gruppen spezifiziert wurde, dennoch eine Standard-Gruppe zugewiesen werden soll. Damit dies zuverlässig funktionieren kann, wurde die Gruppe mit der ID = 1 zwar als mutable, aber als nicht löschbar definiert.

# 15. Nächste Schritte

Der Implementierungsaufwand muss sich im Rahmen dieser Bachelorarbeit einschränken, damit die Deadline eingehalten werden kann. Hier folgen deshalb Vorschläge für mögliche Weiterentwicklungen von Cynget, die aus Zeitmangel nicht mehr einfliessen konnten.

## 15.1. Weitere Policy-Typen, dynamische Typen

Die Liste der Policy Typen ist momentan hard codiert und bindet die Implementationen von Cygnet und strongswan stark aneinander. Die Liste kann auch nicht verändert oder erweitert werden ohne den Code an beiden Orten ändern zu müssen. Sinnvoll wäre daher, die Typen ebenfalls als Tabelle in der Datenbank zu speichern, vom Namen bis hin zum Argument und dessen Format.

## 15.2. Lokalisierung

Mit der Internationalisierung der Applikation ist der Grundstein für die Lokalisierung bereits gelegt. Nun müssen noch die konkreten Übersetzungen geschrieben werden und eine Möglichkeit für den Benutzer, die Sprache umzustellen eingebaut werden.

## 15.3. Bearbeitung von Enforcements in der Policy-Ansicht

Es wäre für die Usability von Vorteil, wenn man die Enforcements direkt in der Policy-Ansicht bearbeiten könnte, beispielsweise mit einer Enforcement-Tabelle pro Policy in der man die 1:n-Beziehung direkt sieht. Der Nutzer kann den mentalen Link zwischen Policy und Gruppe besser machen, wenn er die (soeben erstellte) Policy direkt auf eine oder mehrere Gruppe anwenden kann. Auch für eine spätere Einarbeitung wäre diese Darstellung übersichtlicher.

## 15.4. Unterstützung für längere Sessions

Cygnet geht davon aus, dass das Assessment des Clients innert kurzer Zeit geschieht. Länger dauernde oder kontinuierliche Messungen wie das vergleichbare Produkte von Microsoft bereits implementieren, werden nicht unterstützt. Als Beispiel sei hier eine kontinuierliche Messung genannt, die regelmässig den Status der Firewall überwacht und die VPN-Verbindung trennt, wenn die Firewall deaktiviert wird.

## 15.5. Generierung von Formularen und Auto-Validierung

Die Erzeugung von Formularen und deren Input-Validierung wird von Django vereinfacht angeboten und wurde bei dieser Arbeit nicht eingesetzt. Für zukünftige Versionen von Cygnet wäre es attraktiver, auf die vorhanden Werkzeuge zu setzen anstatt das Rad selber neu zu erfinden.

## 15.6. Action als “Enum”

Durch die unglückliche Definition der Actions im Code benötigt es viel Code, um aus dem numerischen Wert in der Datenbank wieder einen lesbaren String zu machen. Besser wäre das Folgende gewesen, was nicht nur weniger Code benötigt hätte, sondern auch besser internationalisierbar gewesen wäre:

```
1  NONE = 0
2  ALLOW = 1
3  ISOLATE = 2
4  BLOCK = 3
5
6  CHOICES = (
7      (NONE, _('None')),
8      (ALLOW, _('Allow')),
9      (ISOLATE, _('Isolate')),
10     (BLOCK, _('Block')),
11     )
12
13 action = models.IntegerField(choices=CHOICES)
```

Listing 15.1: Action

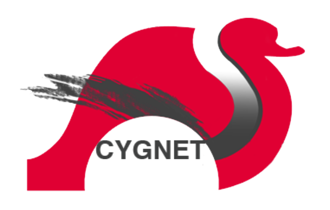






# Cygnet

# Testprotokolle




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 16. Unit-Tests

Automatisierte Unit-Tests sind ein praktisches Werkzeug um festzustellen, ob Änderungen im Code unerwünschte Nebeneffekte haben.

## 16.1. Python

Django liefert bereits ein Skript das Unit-Tests ausführen kann.
Der folgenden Befehl führt alle definierten Unit-Tests aus, die sich im File `cygapp/tests.py` befinden:

```
1 $> python manage.py test cygapp
```

Listing 16.1: Unit Test Befehl

Die Tests werden jedesmal ausgeführt, bevor Änderungen im Code committed, resp. gepushed werden. Wenn weitere Funktionalität hinzukommt, wird der Test-Code entsprechend erweitert.

```
1  /hsr/ba/cygnet$ ./manage.py test cygapp
2  Creating test database for alias 'default'...
3  Creating test database for alias 'meta'...
4  ...........
5  ----------------------------------------------------------------------
6  Ran 11 tests in 0.241s
7
8  OK
9  Destroying test database for alias 'default'...
10 Destroying test database for alias 'meta'...
```

Listing 16.2: Unit Tests Beispielausgabe 29.05.2013

## 16.2. IMV - Simulator

Für die Schnittstellentests mit strongswan wurde ein Python-Skript implementiert, das einen IMV simuliert und Testergebnisse generiert:

```
1  """
2  A module to simulate a StrongSwan IMV used for testing workitem generation
3  and session handling of cygnet. Invoke by calling run_test()
4  """
5
6  import httplib,random
7  from datetime import datetime
8  from models import Session, Device, Identity
9
10 start_url = '/cmd/start_session'
11 end_url   = '/cmd/end_session'
12
13 def start_login(params):
```

```
14      """Call start url to invoke cygnet workItem generation."""
15
16      con = httplib.HTTPConnection('localhost', 8000)
17      url = '%s?%s' % (start_url, '&'.join(['%s=%s' % (k,v) for k,v in
18          params.items()]))
19      con.request('HEAD', url)
20      response = con.getresponse()
21
22      if response.status != 200:
23          raise AssertionError('Expected: HTTP 200, got: HTTP %s' %
24                  response.status)
25
26      body = response.read()
27      if body != '':
28          raise AssertionError('Expceted empty body, got: %s' % body)
29
30
31  def finish_login(params):
32      """Call finish url to invoke cygnet result processing."""
33      con = httplib.HTTPConnection('localhost', 8000)
34
35      url = '%s?%s' % (end_url, '&'.join(['%s=%s' % (k,v) for k,v in
36          params.items()]))
37      con.request('HEAD', url)
38      response = con.getresponse()
39
40      if response.status != 200:
41          raise AssertionError('Expected: HTTP 200, got: HTTP %s' %
42                  response.status)
43
44      body = response.read()
45      if body != '':
46          raise AssertionError('Expceted empty body, got: %s' % body)
47
48  def run_test():
49      """
50      Run the test-case
51      """
52      device = Device.objects.get(value='deadbeef')
53      identity = Identity.objects.get(data='tannerli')
54      session = Session.objects.create(connectionID=random.randint(1,65535), device=device,
55              time=datetime.today(), identity=identity)
56
57      params = {}
58      params['sessionID'] = session.id
59
60      start_login(params)
61
62      #Simulate IMV, generate some random results
63      for item in session.workitems.all():
64          item.error = random.randint(0,1)
65          item.recommendation = random.choice((item.fail, item.noresult))
66          item.result = ''
67          item.save()
68
69      finish_login(params)
70
71      print 'OK'
```

../res/simIMV.py

## 16.3. Selenium

Für das automatisierte Testen der GUI wird das Firefox-Plugin Selenium verwendet, dass Tests einer Website automatisiert ablaufen lässt und eine Reihe von Assert-Befehlen bezüglich des DOM kennt. Auch diese Tests werde regelmässig ausgeführt um Fehler frühmöglichst festzustellen. Die Tests sind im Ordner `selenium` mitgeliefert und können mit dem zugehörigen Plugin durchgeführt werden.

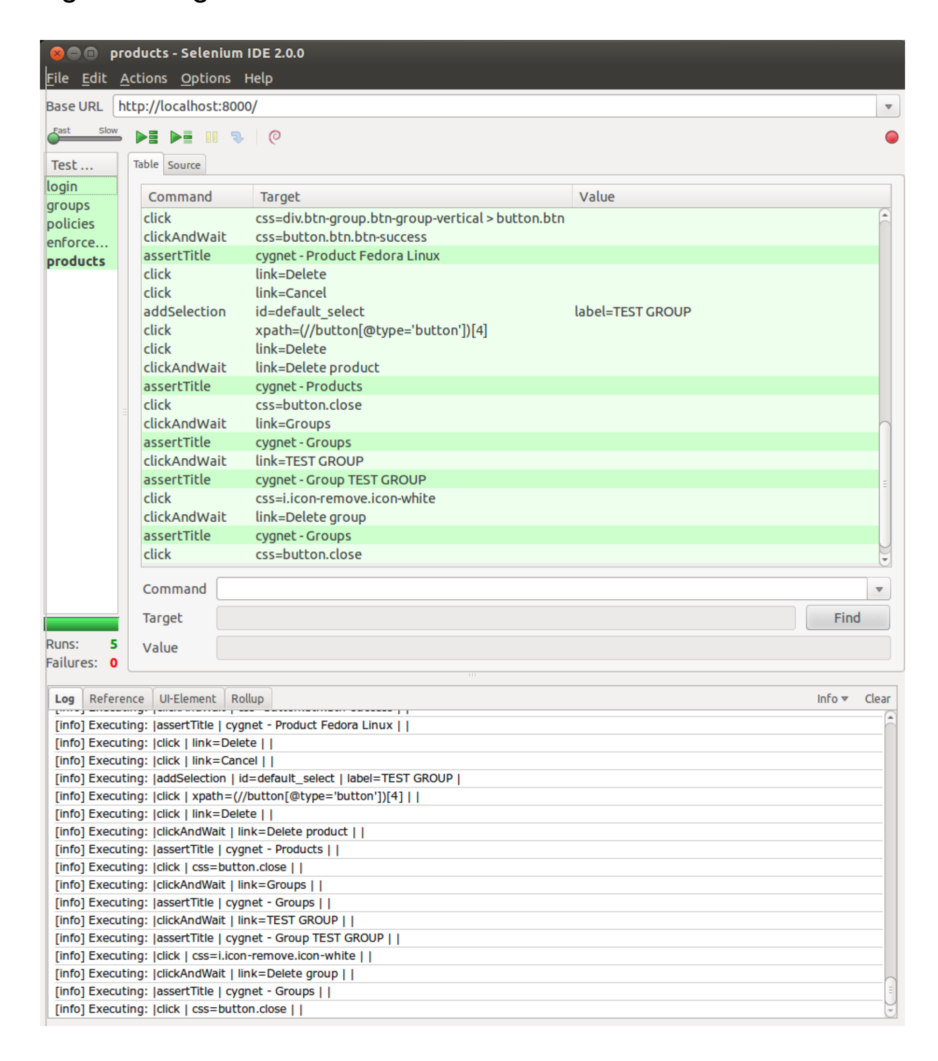



Abbildung 16.1.: Screenshot Selenium

**Hinweis:** *Der Einsatz von Chosen für alle Selektor-Elemente hat das Testen mit Selenium verunmöglicht, die definierten Selenium Tests dienten bis zur letzten Projektwoche als nützliche Hilfe um Bugs zu finden, aber sind jetzt nicht mehr ausführbar.*

# 17. Usecase-Tests

Mit der entwickelten Software sollen alle definierten Usecases durchgeführt werden können. Die Testläufe um den Deckungsgrad festzustellen wurden vom Projektteam durchgeführt. Die Usecases sind im Kapitel 3, Seite 13 definiert.

## 17.1. Testlauf vom 30. Mai 2013

| **Ergebnisse** | |
|---|---|
| **Use Case 0** | Use Case wird vollständig abgedeckt |
| **Use Case 1** | |
| UC1.1 | Use Case wird vollständig abgedeckt |
| UC1.2 | Use Case wird vollständig abgedeckt |
| UC1.3 | Use Case wird vollständig abgedeckt |
| **Use Case 2** | Use Case wird nicht abgedeckt, siehe 17.1.1 |

Tabelle 17.1.: UC-Test Ergebnisse

### 17.1.1. Resultat Use Case 2

Der Usecase definiert, dass die letzten Prüfungen eines Geräts eingesehen werden können sollen. Dies ist momentan nicht der Fall, es können nur die allerletzten Ergebnisse betrachtet werden.

**Massnahme:**
Report-Ansicht erweitern. Siehe
`https://github.com/tannerli/cygnet-doc/issues/83`

## 17.2. Testlauf vom 03. Juni 2013

| **Ergebnisse** | |
|---|---|
| **Use Case 0** | Use Case wird vollständig abgedeckt |
| **Use Case 1** | |
| UC1.1 | Use Case wird vollständig abgedeckt |
| UC1.2 | Use Case wird vollständig abgedeckt |
| UC1.3 | Use Case wird vollständig abgedeckt |
| **Use Case 2** | Use Case wird vollständig abgedeckt |

Tabelle 17.2.: UC-Test Ergebnisse 2

Alle Use-Cases werden abgedeckt. Die Anforderungen der Use-Cases sind somit erfüllt.

# 18. Usability-Tests

Der folgende Test soll sicherstellen, dass das Benutzerinterface selbsterklärend und verständlich ist, ohne ein vorhandenes Wissen über die Problemdomäne. Die Instruktionen sind deshalb entsprechend detailliert verfasst und es steht ein Experte für Rückfragen zur Verfügung.

## 18.1. Instruktionen

### 18.1.1. Einleitung

Sie sind ein Informatik-Systemadministrator der ein Computernetzwerk verwaltet. Für diese Aufgabe stehen Ihnen die folgenden Hilfsmittel zur Verfügung:

- Ein Internet-Browser mit Cygnet (Verwaltungssoftware)
- Das Cygnet Benutzerhandbuch als PDF
- Diese Instruktionen

Bitte bearbeiten Sie die folgenden Anweisungen nach bestem Wissen und Gewissen, wenn möglich selbstständig. Wenn Sie Fragen zum Test-Ablauf haben, stellen Sie diese bitte bevor sie fortfahren.
Sprechen Sie während dem Arbeiten Ihre Gedankengänge und Aktionen laut aus, d.h. kommentieren Sie was Sie tun, und warum. Beantworten Sie die in den Instruktionen gestellten Fragen dem Betreuer.

### 18.1.2. Arbeitsanweisungen

Der Internetbrowser ist bereits auf der Cygnet-Startseite geöffnet. Loggen Sie sich ein. Das Passwort lautet '`password`'.
Navigieren Sie zur Gruppenverwaltung.
Wie viele Gruppen gibt es momentan?
Wie viele Mitglieder (Geräte) hat die Gruppe “Admins”?
Welches ist die übergeordnete Gruppe der Gruppe “Consultants”?

Sie möchten eine neue Gruppe anlegen. Nennen Sie sie “Customer Support”. Weisen Sie der Gruppe alle verfügbaren Geräte als Mitglieder zu und speichern Sie die Gruppe ab.

Versuchen Sie herauszufinden, ob sich das Software-Paket “passwd” zur Zeit auf der schwarzen Liste (Blacklist) befindet.

Weisen Sie das Paket “firefox” der schwarzen Liste zu.

Finden Sie heraus, welches Betriebssystem (Produkt) auf dem Gerät “Sales Persons Laptop” installiert ist.

Löschen Sie die Datei (File) “/bin/ping6”.

Legen Sie eine neue Policy an. Nennen Sie sie “Exist cat”. Die Policy soll testen, ob eine bestimmte Datei existiert (Tipp: FileExist). Die Datei die Sie testen möchten heisst “/bin/cat”. Wenn die Policy vom Client (Gerät) nicht erfüllt wird, also fehlschlägt, soll er geblockt werden. Wenn die Messung nicht durchgeführt werden kann oder kein Resultat vorhanden ist, soll der Client isoliert werden. Speichern Sie die Policy.

Erstellen Sie ein neues Enforcement, das die zuvor erstellte Policy mit der Gruppe “Customer Support” verbindet. Sie möchten, dass diese Policy mindestens alle 5 Tage getestet wird. Die Aktionen bei einem Fehlschlag oder wenn kein Resultat vorhanden ist, sollen von der Policy übernommen werden. Speichern Sie das Enforcement.

Finden und betrachten Sie den Gerätebericht des Geräts mit der ID “a654e8da64”.
Was ist das letzte Resultat der von Ihnen erstellten Policy (Exist cat)?
Wird diese Policy beim nächsten Login getestet?

Loggen Sie sich aus.

## 18.2. Testlauf vom 01. Juni 2013

**Testperson:** Seraina Pfyl

**Ausbildung:** Kaufmännische Berufslehre HGT

**Englischkenntnisse:** First Certificate (B2)

**Stand der Software:** Commit `aa52f85` [1]

**Stand der Doku:** Commit `c665e51` [2]

**Instruktionen:** Schriftlich, siehe 18.1

### 18.2.1. Ergebnisse

Die Instruktionen wurden verstanden und grundsätzlich erfolgreich ausgeführt. Folgende Schwierigkeiten traten während der Bearbeitung auf:

- Bei der Gruppenübersicht ist nicht von Anfang an klar, wieviele Gruppen z.Z. existieren. Nur die beiden Top-Level-Gruppen wurden als solche erkannt.
- Zuordnungsfeld Mitglieder zu Gruppe oder Gruppe zu Gerät ist nur dann klar, wenn die beiden Felder nebeneinander angeordnet sind. Ist das Fenster zu schmal ist es unklar.
- Beim Zuordnungsfeld ist unklar, welches Feld die Mitglieder und welches die vorhandenen Elemente darstellt.
- Das Suchfeld kann mit Dateipfaden nicht umgehen. Eine Suche nach “/bin/ping6” funktioniert nicht.
- Beim Ändern einer Blacklist-Einstellung für ein Paket wird der Modal-Dialog nach kurzer Zeit automatisch ausgeblendet.
- Der Link zum “Device Report” ist zu klein.

Das allgemeine Feedback der Testperson war im Grundton positiv:

> *Das Design ist konsistent, das Plus-Symbol für ein neues Element ist sinnvoll und das Plus als Zeichen allgemein bekannt. Es ist gut, dass bei Unsicherheit ein Tooltip existiert. Die Overview-Ansicht ist cool. Das Design der Seite ist ansprechend und die Farbverwendung (blau) ist freundlich. Die Farben bei Steuerelementen (rot/grün für Buttons) sind passend und hilfreich. Insgesamt wirken die Seiten allerdings ein wenig leer, ausser die Übersicht.*

[1] `https://github.com/tannerli/cygnet/commit/aa52f8516c467568b33b39be6b5899585145896b`
[2] `https://github.com/tannerli/cygnet-doc/commit/c665e5192675b39da28034b441ea6ff9e6f1c4ec`

## 18.3. Testlauf vom 06. Juni 2013

| | |
|---|---|
| **Testperson:** | Urs Baumann |
| **Ausbildung:** | Informatiker (HSR Bsc. Student) |
| **Englischkenntnisse:** | HSR-Englischmodul 3 (ca. C1) |
| **Stand der Software:** | Commit `fdec327` [3] |
| **Stand der Doku:** | Commit `6f0f94b` [4] |
| **Instruktionen:** | Schriftlich, siehe 18.1 |

### 18.3.1. Ergebnisse

Die Instruktionen konnten ohne Schwierigkeiten bearbeitet werden. Einzig der Wunsch nach Tooltips bei den zwei Zuweisungsbuttons in der Gruppenverwaltung kam auf.

Herr Baumann hat auf ein persönliches Statement verzichtet.

[3] https://github.com/tannerli/cygnet/commit/fdec3278ecb56d3c7a4bc3d626c2ae9f55273bc1
[4] https://github.com/tannerli/cygnet-doc/commit/6f0f94b54c223fa4c9e4c960786d93b6f5a1bd33

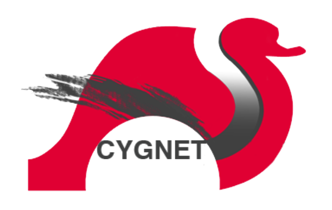






# Cygnet

# Deployment Manual




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 19. Vorgehensweise

Dieses Dokument beschreibt die Vorgehensweise um die Cygnet-Applikation auf einem Webserver in Betrieb zu nehmen. Die Anleitung wurde mit einem Ubuntu Server 13.04 "Raring Ringtail" getestet. Der Einsatz von Cygnet auf ähnlichen Linux-Systemen sollte kein Problem darstellen.

## 19.1. Server

Es wird ein Server benötigt auf dem folgende Software verfügbar ist:

- apache2
- python
- python-django
- libapache2-mod-wsgi

Um die Software zu installieren kann auf einem Terminal der folgende Befehl verwendet werden:

```
$> sudo apt-get install apache2 python python-django libapache2-mod-wsgi
```

Listing 19.1: apt-get

**Hinweis:** *Es wird mindestens Django Version 1.5 benötigt. Diese ist bei Ubuntu 13.04 enthalten. Wenn ein älteres Ubuntu wie 12.10 verwendet wird muss Django noch manuell aktualisiert werden. Siehe offizielle Django-Dokumentation[1] für eine Anleitung*

## 19.2. Cygnet

Für die "Installation" von Cygnet reicht es, das Archiv `cygnet.zip` an eine geeignete Stelle (z.B. `/var/www`) zu entpacken, und den Dateibesitzer auf den Apache-User zu setzen:

```
$> unzip cygnet.zip
$> cp -r cygnet/ /var/www
$> cd !$
$> sudo chown -R www-data:www-data cygnet
```

Listing 19.2: Copy files

[1] https://docs.djangoproject.com/en/dev/topics/install/

### 19.2.1. Konfiguration Datenbank

Cygnet wird mit zwei Datenbankdateien (SQLite) ausgeliefert: `ipsec.config.db` und `django.db`, beide im Root-Verzeichnis von `cygnet.zip`.

Der Pfad zu den Datenbanken muss in der Datei `/var/www/cygnet/settings.py` angepasst werden, dazu muss der `NAME`-Wert richtig gesetzt werden, es werden keine relativen Pfadangaben unterstützt:

```
# settings.py

DATABASES = {

        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': '/var/www/cygnet/ipsec.config.db',
            },

        'meta': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': '/var/www/cygnet/django.db',
            },
        }
```

Listing 19.3: Databases

### 19.2.2. Passwort

Das Webinterface von Cygnet ist mit einem Passwort geschützt. Dieses Passwort muss erstmalig gesetzt werden. Dies kann mit dem folgenden Befehl gemacht werden, wobei `meinPasswort` durch das effektive Passwort zu ersetzen ist:

```
$> cd /var/www/cygnet
$> python manage.py setpassword meinPasswort
```

Listing 19.4: Set password I

Wenn das Passwort nicht auf dem Terminal dargestellt werden soll, kann das Passwort-Argument weggelassen werden. Es wird interaktiv erfragt und auf der Konsole nicht ausgegeben:

```
$> cd /var/www/cygnet
$> python manage.py setpassword
looking for cygnet user in database...
please enter a new password for cygnet-user:
password updated successfully
$>
```

Listing 19.5: Set password II

Dieser Befehl kann auch nachträglich verwendet werden, um das Passwort zurückzusetzen, falls es vergessen wurde.

## 19.3. Konfiguration Apache

Nun muss die Apache Konfiguration angepasst werden. Davon ausgehend, dass Cygnet die einzige Web-Applikation auf dem Server ist, kann das in der Datei `/etc/apache2/sites-available/default` gemacht werden:

```
WSGIScriptAlias / /var/www/cygnet/wsgi.py
WSGIPythonPath /var/www/cygnet

Alias /static/ /var/www/cygnet/cygapp/static/

<VirtualHost *:80>
    DocumentRoot /var/www/cygnet

    <Directory /var/www/cygnet>
        <Files wsgi.py>
            Order deny,allow
            Allow from all
        </Files>
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Listing 19.6: Apache configuration

Um die Konfiguration zu aktivieren, muss der Webserver neu gestartet werden:

```
$> sudo service apache2 restart
```

Listing 19.7: Apache restart

Cygnet sollte nun für einen Webbrowser unter `http://localhost/` erreichbar sein.

## 19.4. (Optional) IMV Authentisierung

**Warnung zu Sicherheitsinstruktionen:** *Die folgenden Konfigurationen gehen von einem Standardsystem aus und sind möglicherweise unvollständig oder veraltet. Die nötigen Konfigurationen können auf Ihrem System abweichen. Bitte gehen Sie vorsichtig vor und konsultieren Sie die Apache-Dokumentation.*

Damit nur ein IMV eine Cygnet-Session starten bzw. beenden kann, muss der Zugriff auf die URLs `/cmd/*` beschränkt werden.
Da für das Projekt eine SQlite-Datenbank verwendet wurde, ist davon auszugehen, das sowohl Cygnet als auch alle IMVs auf ein und derselben Maschine ausgeführt werden. Es reicht daher aus, den Zugriff auf die URLs nur von localhost zu erlauben. Das kann mit einer zusätzlichen Direktive in der Apache-Konfiguration realisiert werden (innerhalb des bestehenden VirtualHosts):

```
<Location /cmd>
    Order deny,allow
    allow from 127.0.0.1
    deny from all
</Location>
```

Listing 19.8: IMV Auth

Falls wider Erwarten der Zugriff des IMVs über eine Netzwerkverbindung geschieht, könnte die Authentisierung mit Apache Basic Authentication realisiert werden. Die zugehörige Konfiguration würde dann etwa so aussehen:

```
<Location /cmd>
    AuthType Basic
    AuthName "IMV API Secion"
    AuthUserFile "/path/to/.htpasswd"
    Require valid-user

    Order allow,deny
    allow from all
</Location>
```

Listing 19.9: IMV Basic Auth

Das `htpasswd`-File kann mit folgendem Befehl erstellt werden. Der Benutzer imv_auth wird direkt angelegt. Ein Passwort wird interaktiv erfragt.

```
$> htpasswd -c /path/to/.htpasswd imv_auth
```

Listing 19.10: Htpasswd

Der IMV kann sich dann mit dem Benutzernamen imv_auth und dem angegebenen Passwort authentisieren.

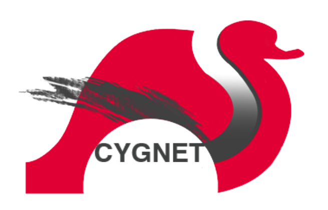






# Cygnet

# User Manual




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 20. Einführung

Dieses Dokument beschreibt die Funktionsweise und Bedienmöglichkeiten von Cygnet. Es ist an den Enduser gerichtet.

## 20.1. Was ist Cygnet?

Cygnet ist eine Erweiterung für den strongSwan VPN-Client und den dazugehörigen Server. Es ermöglicht die Definition und Durchsetzung von Richtlinien, die für alle VPN-Clients gelten und bei einem Verbindungsversuch erfüllt werden müssen.

StrongSwan-VPN-Clients (**Clients**) können in Cygnet in **Gruppen** eingeteilt werden, nach Betriebssystem, Unternehmensstruktur oder persönlicher Präferenz des Administrators. Es können Richtlinien (**Policies**) definiert werden, wie beispielsweise, dass alle verfügbaren Betriebssystemupdates auf dem Client installiert sind, oder dass gewisse Applikationen auf dem Client nicht installiert sein dürfen. Diese Richtlinien können dann auf die Gruppen angewandt/erzwungen (**Enforcements**) werden und werden von Cygnet bei einem Verbindungsversuch eines Clients geprüft.

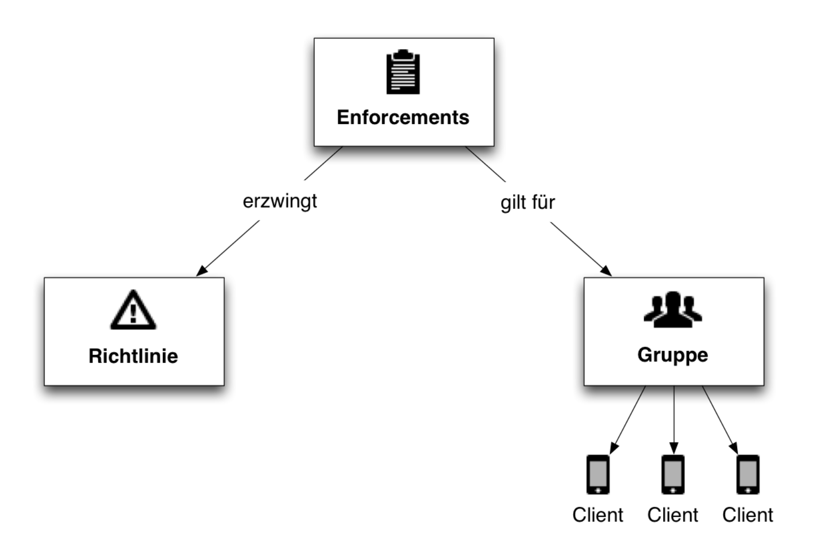



Abbildung 20.1.: Übersicht

# 21. Schritt für Schritt Anleitung

In diesem Kapitel wird anhand eines Beispiels die Einrichtung einer VPN-Richtlinie für eine Unternehmung gezeigt.

## 21.1. Ausgangslage

Es wird davon ausgegangen, dass bereits eine strongSwan-Installation mit Cygnet existiert. Siehe 19 Deployment.
Als Beispiel sei eine Hochschule, die einen VPN-Zugang für Studenten, Dozenten und Mitarbeiter anbietet. Jeder dieser Benutzer hat bereits einen Benutzernamen und ein Passwort erhalten, welche als Zugang für das VPN genügen. Grundsätzlich kann sich jeder Benutzer mit einem beliebigen (auch privaten) VPN-fähigen Gerät ins Hochschulnetz einwählen. Damit die Sicherheit des internen Hochschulnetzes nicht kompromittiert wird, wird mithilfe von Cygnet eine Sicherheitsrichtlinie definiert und durchgesetzt.

Wenn Sie Cygnet in einem Browser öffnen und Ihr Passwort eingeben, erscheint als erstes die Übersicht:

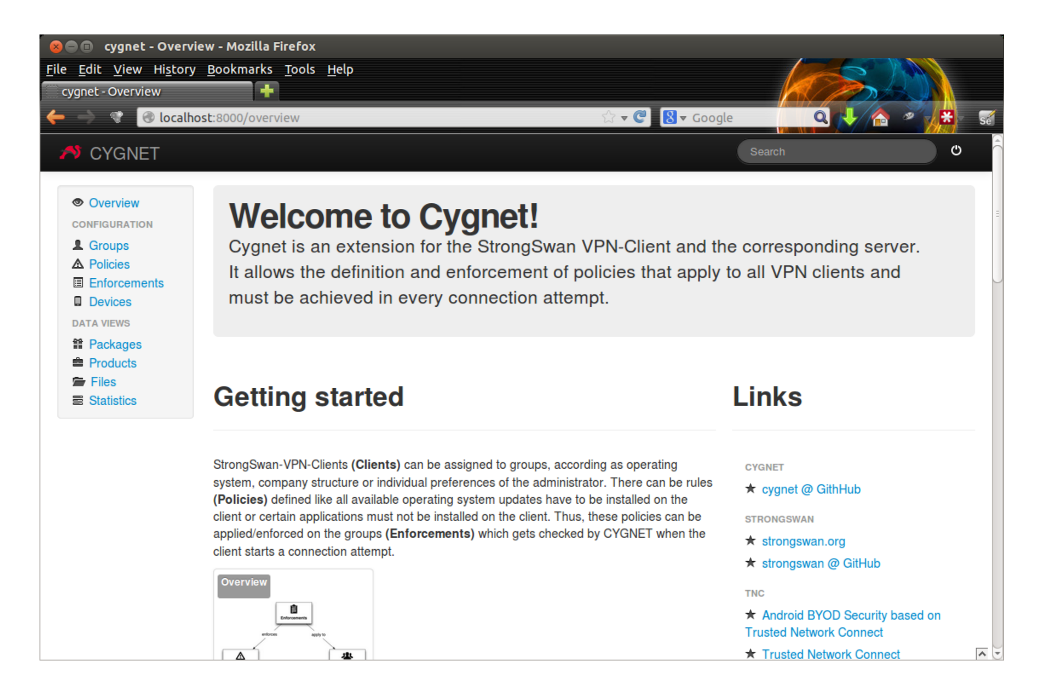



Abbildung 21.1.: Cygnet: Übersicht

## 21.2. Gruppen

Über den ersten Navigationspunkt gelangen Sie auf die Gruppenübersicht. Anfangs sind noch keine Gruppen definiert. Neue Gruppen können mit dem “Hinzufügen”-Button (blaues Plus-Zeichen) erstellt werden.

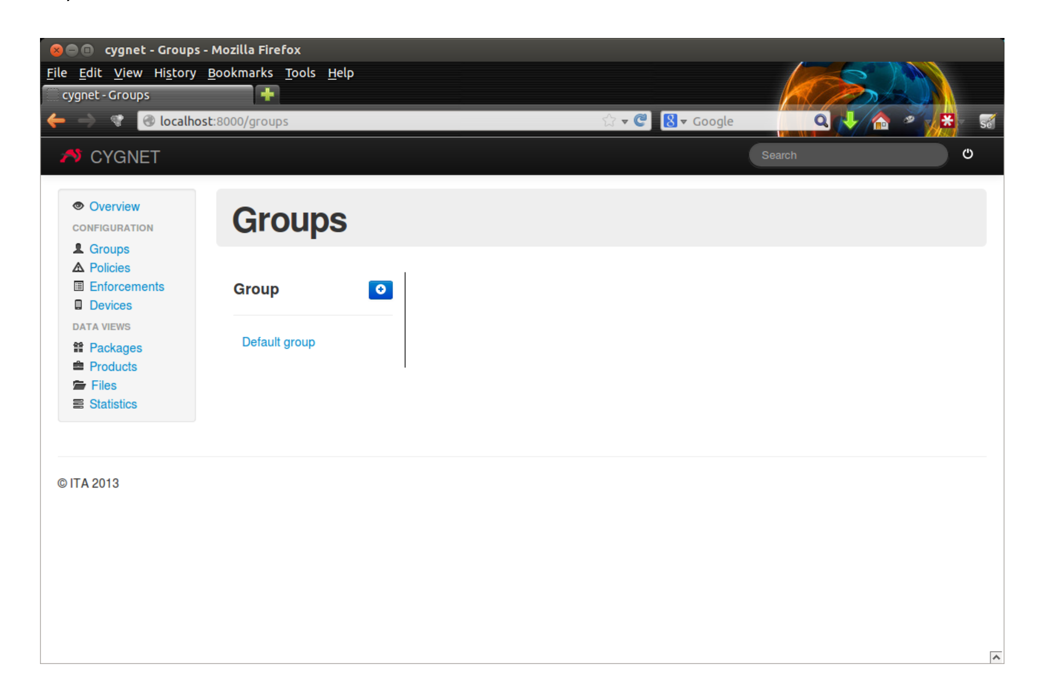



Abbildung 21.2.: Cygnet: Gruppen

Eine Gruppe ist definiert über Ihren Namen und kann eine übergeordnete Gruppe haben. Ein Client, der Mitglied in einer Gruppe ist, ist automatisch auch Mitglied in allen übergeordneten Gruppen der Gruppe. Für das Beispiel werden einzelne Gruppen für Studenten, Dozenten und Mitarbeiter definiert, sowie passende Untergruppen. Wenn also ein Client Mitglied der Gruppe “Assistenten” ist, gelten für den Client automatisch auch alle Enforcements der Gruppe “Dozenten”.

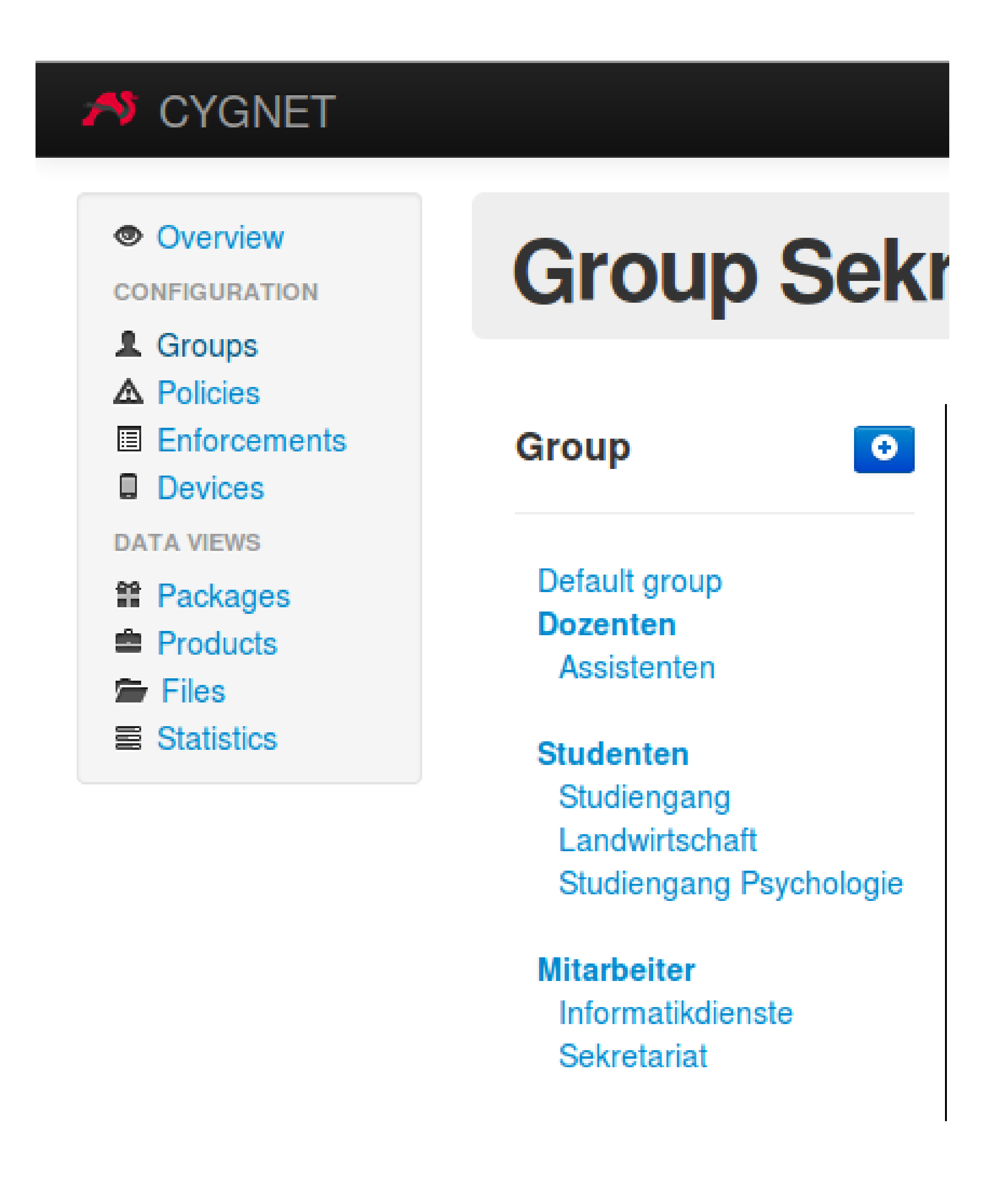



Abbildung 21.3.: Cygnet: Beispielgruppen

## 21.3. Richtlinien (Policies)

Im nächsten Schritt sollen Richtlinien für die verschiedenen Gruppen erstellt werden. Dafür kann im Navigationspunkt “Policies” eine neue Richtlinie erstellt werden.

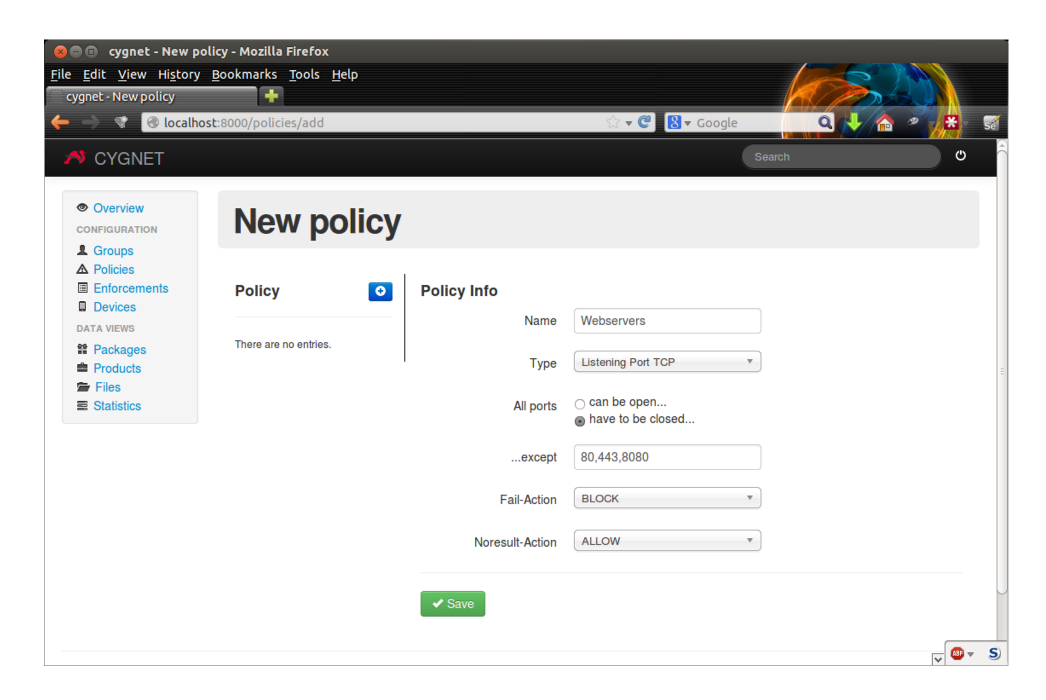



Abbildung 21.4.: Cygnet: Neue Policy

Eine Policy besteht aus folgenden Eigenschaften:

| Eigenschaft | Beschreibung |
|---|---|
| Name | Name der Policy |
| Typ | Typ der Policy (siehe 21.3.1 für mehr Informationen) |
| Argument | Hängt vom Typ ab. Siehe 21.3.1 |
| Fail-Aktion | Legt fest, wie das System reagiert, wenn die Richtlinie vom Client nicht erfüllt wird. |
| Noresult-Aktion | Legt fest, wie das System reagiert, wenn eine Messung auf einem Client nicht durchgeführt werden kann. |

Beide dieser Aktions-Felder haben vier Auswahlmöglichkeiten. Wenn die Bedingungen der Richtlinie erfüllt werden, ist die Empfehlung immer “ALLOW”.

| Aktion | Bedeutung |
|---|---|
| NONE | Keine Empfehlung |
| ALLOW | Erlauben |
| ISOLATE | Isolieren, in ein Spezialnetz verbinden |
| BLOCK | Blockieren, Verbindung verweigern |

Wenn mehrere Richtlinien mit unterschiedlichen Ergebnissen auf einen Client angewandt werden, zählt immer das “schlechtere” Ergebnis, also BLOCK vor ISOLATE vor ALLOW.

### 21.3.1. Policy-Typen

Die folgenden Policy-Typen werden zurzeit von Cygnet unterstützt.

### File Hash

Diese Policy prüft, ob der Hash einer Datei auf dem Client mit dem entsprechenden Referenzwert in der Datenbank übereinstimmt. Die Datei kann aus einer Auswahlliste ausgewählt werden. Der passende Referenzwert wird anhand des Betriebssystems des Clients bestimmt.

### Dir Hash

Wie File Hash, aber prüft sämtliche bekannten Dateien in einem Verzeichnis.

### Listening Port TCP/UDP

Die Policy prüft, ob auf den angegebenen Ports Listening Sockets eröffnet wurden. Die Port-Range kann beispielsweise folgendermassen aussehen:
`21, 22, 80, 443, 1000-1500, 2048, 6000-40000`

### File Exist

Die Policy prüft, ob eine bestimmte Datei auf dem Client existiert. Sie schlägt fehl, wenn die Datei nicht existiert.

### File Not Exist

Wie File Exist, aber prüft, ob die Datei NICHT existiert und schlägt fehl, wenn die Datei existiert.

### Missing Update

Prüft, ob alle installierten Softwarepakete (etwa aus dem Google Play Store) auf dem aktuellsten Stand sind. Schlägt fehl, wenn ein Update fehlt.

### Missing Security Update

Wie Missing Update, prüft aber nur auf sicherheitsrelevante Updates. Nicht sicherheitsrelevante Updates dürfen fehlen und die Policy ist trotzdem erfolgreich.

### Blacklisted Package

Prüft, ob ein Softwarepaket installiert wurde, das vom Administrator auf die Blacklist gesetzt wurde.

### OSSettings

Der IMV kontrolliert, ob gewisse Betriebssystemoptionen korrekt gesetzt sind. Ist von der strongSwan-Implementation abhängig.

### Deny

Diese Policy prüft nichts. Sie schlägt per Definition fehl und dient hauptsächlich zum definieren einer Geräte-Blacklist.

## 21.4. Enforcements

Als dritter und letzter Schritt müssen die erstellen Gruppen und Policies einander zugeordnet werden. Dies kann im Navigationspunkt “Enforcements” gemacht werden. Ein Enforcement setzt eine Policy auf einer Gruppe um.

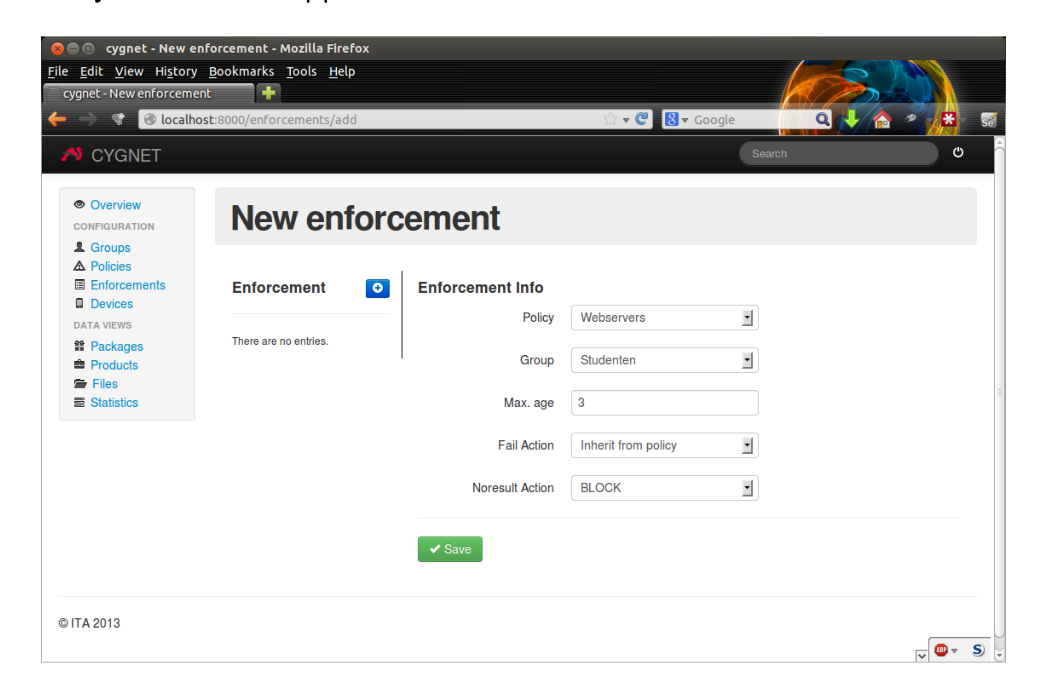



Abbildung 21.5.: Cygnet: Neues Enforcement

Zusätzlich kann ein Zeitintervall in Tagen angegeben werden, das angibt, wie oft die Policy getestet werden soll. Ein Wert von 3 bedeutet, dass die Policy, wenn die letzte Prüfung erfolgreich war, nur alle 3 Tage geprüft wird. Ein Wert von 0 bedeutet, dass die Policy jedes Mal geprüft wird.
Bei einem Enforcement können die “Fail-Action” und “Noresult-Action” überschrieben werden, falls gewünscht. Standardmässig werden die Aktionen von der gewählten Policy geerbt. Wenn ein Client in zwei Gruppen eingeteilt ist auf denen dieselbe Policy angewandt wurde, so wird die Policy nur einmal getestet. Wenn sich die konfigurierten Aktionen oder das Zeitintervall unterscheiden so wird die drastischere Aktion (BLOCK vor ISOLATE vor ALLOW), resp. das kürzere Zeitintervall angewandt.

## 21.5. Geräte / Clients

In diesem Bereich werden allen bekannten Clients aufgelistet. Die Liste wird automatisch um neue Clients ergänzt. Ein Client kann hier Gruppen zugeordnet werden, auf denen allenfalls

weitere Policies angewandt wurden. Es kann ein Beschreibungstext zum Gerät zugeordnet werden, um die Wiedererkennbarkeit zu vereinfachen.

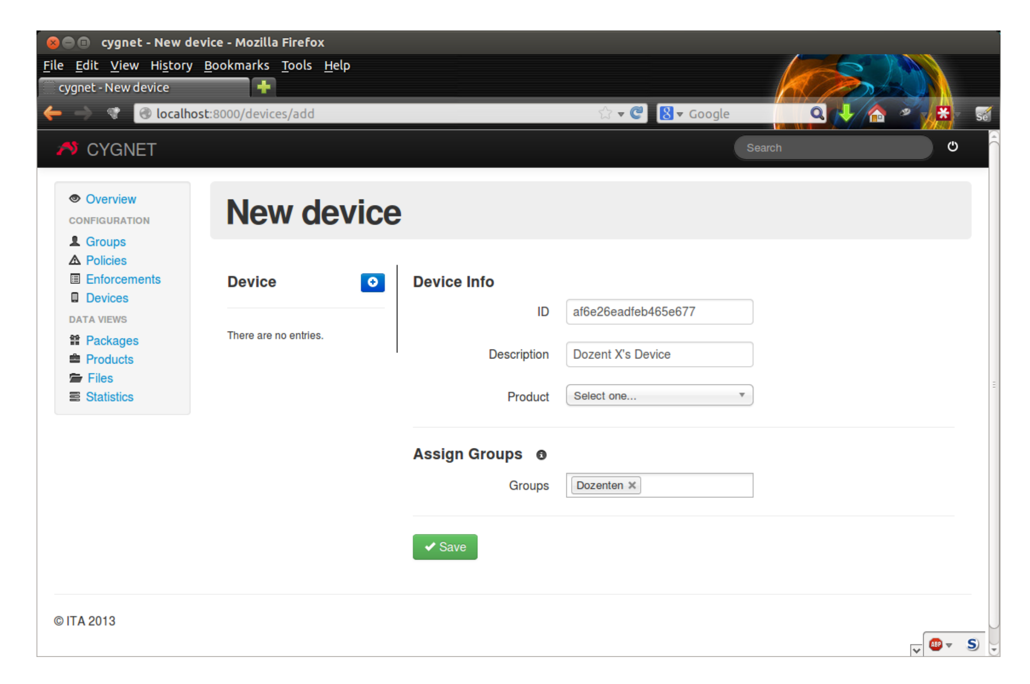



Abbildung 21.6.: Cygnet: Neues Gerät

Zusätzlich kann hier der "Device-Report" eingesehen werden. In dieser Ansicht werden die letzten Messergebnisse eines Geräts angezeigt, in welchen Gruppen das Gerät eingeteilt ist und welche Enforcements auf das Gerät wirken. Dies kann nützlich sein, um herauszufinden warum ein Gerät blockiert oder isoliert wurde.

## 21.6. Dateien / Files

In diesem Abschnitt werden die bekannten Dateihashes gespeichert. Sie dienen als Referenzwerte für File Hash und Dir Hash Richtlinien. Dateien und Hashes sind read-only. Sie können nur gelöscht, aber nicht bearbeitet oder neu erfasst werden. Die zurzeit bekannten und unterstützten Hash-Algorithmen sind:

- SHA-1
- SHA-1-IMA
- SHA-256
- SHA-384

## 21.7. Pakete / Packages

Auf dieser Seite werden alle bekannten Software-Pakete und deren Versionen aufgelistet. Hier kann definiert werden, welche Pakete auf der Blacklist stehen und für die "Blacklisted Package"-Policy (Seite 69) getestet werden.

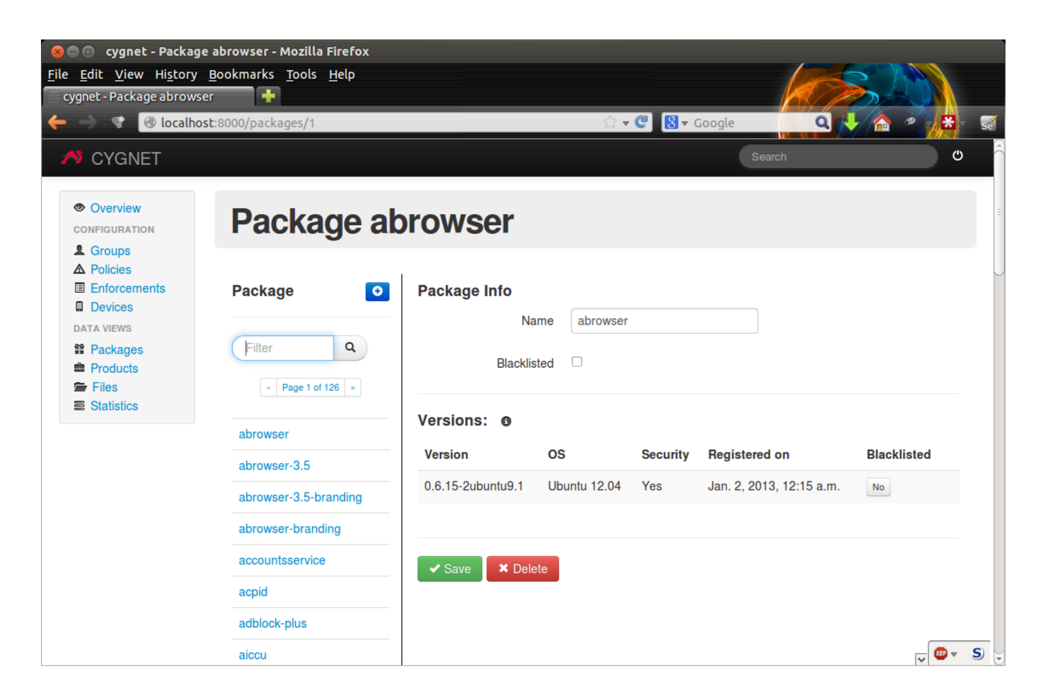



Abbildung 21.7.: Cygnet: Neues Paket

Ein Paket kann entweder global blockiert werden oder nur einzelne Versionen davon. Wenn die Einstellung global für das Paket verändert wird, werden die allenfalls gemachten Konfigurationen der einzelnen Versionen überschrieben.

## 21.8. Produkte / Products

Die hier aufgelisteten Produkte sind alle Client-Betriebssysteme die bisher bei Clients installiert waren. Die Liste wird automatisch ergänzt, wenn Clients mit einem anderen Betriebssystem auftauchen. In diesem Bereich können Standard-Gruppen zu Betriebssystemen zugeordnet werden. Das bedeutet, dass wenn ein neuer Client mit einem bestimmten Betriebssystem zum ersten Mal eine Verbindung aufbaut, werden ihm automatisch die Standard-Gruppen des Produkts zugeordnet. So kann eine Default-Policy für unterschiedliche Betriebssysteme konfiguriert werden.
Wenn ein Client ein bisher unbekanntes Betriebssystem installiert hat, wird er in die Gruppe “Default group” hinzugefügt.

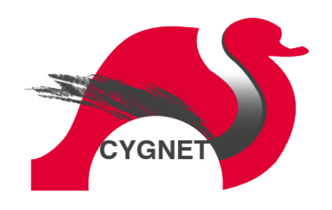






# Cygnet

# Projektplan




**Bachelorarbeit FS2013**

**Studenten:** Stefan Rohner, Marco Tanner

**Betreuer:** Prof. Dr. Andreas Steffen, Tobias Brunner

**Gegenleser:** Prof. Stefan Keller

**Experte:** Dr. Ralf Hauser

# 22. Einführung

## 22.1. Zweck

In diesem Dokument werden die Details zur Bachelorarbeit “BYOD Malware Database Management Tool for Android” dargelegt und die Projektmeilensteine definiert.

## 22.2. Aufgabenstellung

### 22.2.1. Beschreibung

Die neue BYOD (Bring-Your-Own-Device) Version des populären strongSwan Android VPN Clients erlaubt es auf der Basis der standardisierten Trusted Network Connect (TNC) Protokolle den Gesundheitszustand von Android 4 Smartphones und Tablets remote zu bestimmen. Zur Zeit können offene Serverports erkannt, eine Liste aller installierten Apps erfasst, die SHA-1 Checksummen von System-Programmen und Libraries berechnet und gewisse Geräteeinstellungen überwacht werden.
Auf der TNC Serverseite wird zur Zeit eine Blacklist von potentiell gefährlichen Server-Apps in einer SQLite Datenbank gespeichert, die über ein Kommandozeilentool rudimentär verwaltet werden kann. Clients können über diese Liste auch zu einen Security-Update auf eine neuere Version gezwungen werden. Diese Liste soll jederzeit entweder manuell oder eventuell durch Anflanschen von Internet-basierten Datenbanken aktuell gehalten werden können. Auch sollen Hash-Referenzwerte von verschiedenen Versionen der Android System Software verwaltet werden können. Weiter soll über Policies eingestellt werden können, welche Gesundheitstests beim Login in ein Firmennetz auf der Basis des früheren Verhaltens jeweils von einem spezifischen Android Gerät verlangt werden sollen.
Dafür soll ein Web-basiertes Management Tool entwickelt werden, das zusammen mit einem Apache Webserver auf einer Linux Plattform laufen soll. Die Programmier- oder Skriptsprache für diese Serveranwendung ist frei wählbar, unter der Bedingung, dass nur Open Source Software Komponenten verwendet werden.

### 22.2.2. Ziele

- Einarbeiten in die Funktionalität des Android BYOD Clients und in die Trusted Network Connect Architektur der Trusted Computing Group.
- Wahl eines geeigneten Open Source Web-Server-Framework.
- Definition der BYOD Policy Rule Datenbankschnittstelle.

- Spezifikation, Implementation und Test der Android BYOD Security Management Tool Funktionalität.

### 22.2.3. Links

- Christoph Bühler & Patrick Lötscher, strongSwan Android 4 Client with Endpoint Assessment“, HSR Studienarbeit HS12
  http://security.hsr.ch/projects/SA_2012_strongSwan-Android4-Client-with-Endpoint-Assessment.pdf
- Android BYOD Security based on Trusted Network Connect
  http://wiki.strongswan.org/projects/strongswan/wiki/BYOD
- Trusted Network Connect Architecture
  http://wiki.strongswan.org/projects/strongswan/wiki/TrustedNetworkConnect

## 22.3. Abgabetermin

Der späteste Abgabetermin ist der Freitag, 14. Juni 2013.

# 23. Projektorganisation

## 23.1. Team

Das Projekt wird von Stefan Rohner und Marco Tanner bearbeitet. Die Projektleitung wird im Team abgewickelt.

## 23.2. Externe Kontakte

Der Auftraggeber des Projekts ist das HSR Institut für Internet-Technologien und Anwendungen, namentlich Prof. Dr. Andreas Steffen und Tobias Brunner.

## 23.3. Besprechungen

Jeweils am Montag um 13:30 treffen sich das Projektteam und die Betreuer zu einem wöchentlichen Statusupdate. (Raum 6.110, ITA)

## 23.4. Arbeitsumfang

Die Bachelorarbeit ist als 12-Punkte Modul definiert, das bedeutet zwölf mal 25 bis 30 Arbeitsstunden pro Student. Das heisst der Umfang der Arbeit sollte zwischen 600 und 720 Arbeitsstunden betragen.

## 23.5. Arbeitsumgebung

### 23.5.1. Infrastruktur

Zur Projektbewältigung arbeitet jedes Teammitglied mit seinem persönlichen Laptop, sowie dem von der HSR zur Verfügung gestellten Desktop-Rechner. Für die Entwicklung des Management-Tools und die Durchführung von realitätsnahen Systemtests, steht ein Google Nexus 7 Tablet zur Verfügung, welches vom ITA gestellt wurde.
Zur Sicherstellung der Versionskontrolle sowie die Dokumentenverwaltung und das Projektmanagement wird Git auf Github verwendet. Die Zeiterfassung wird von jedem Mitarbeiter selbständig mitgeführt.

### 23.5.2. Tools

Folgende Software findet Einsatz in diesem Projekt:

- **Django & Python 2.7**
  dient als Grundlage für das geplante Tool.
- **git**
  wird als Versionierungssystem eingesetzt.
- **Github.com**
  hostet unser Git-Repository (als private-repo, nicht öffentlich zugänglich).
- **LaTeX**
  wird für das Setzen der Dokumentation verwendet.
- **astah***
  für das Erstellen von UML-Diagrammen.
- **Dropbox**
  für den kurzfristigen Austausch von Dateien.

## 23.6. Qualitätsmassnahmen

### 23.6.1. Projektmanagement-Tool

Es wird die auf Github zur Verfügung gestellten Issues-Funktionen für das Projektmanagement verwendet. Dies erlaubt Meilensteine, auf welche Aufgaben zugewiesen sind, zu definieren. Diese Aufgaben können Mitarbeitern zur Bearbeitung vergeben werden.

### 23.6.2. Dokumentation

Erstellte Dokumente werden jeweils vom anderen Teammitglied gegengelesen, korrigiert und überarbeitet. Zudem sind Dokumenten-Reviews im Projektplan ersichtlich, bei welchen die Dokumentation zusammen mit dem Betreuer besprochen wird.

### 23.6.3. Code-Richtlinien

Es ist insbesondere für Whitespace-strukturierte Sprachen wie Python enorm wichtig, dass gewisse Style-Guidelines eingehalten werden, damit der Code von allen Entwicklern reibungsfrei zusammenarbeitet. Um die Codequalität hoch zu halten, soll sich sämtlicher Python-Sourcecode an die PEP (Python Enhancement Proposal) 8 Richtlinien [1] halten. Diese haben sich in der Community etabliert und bewährt. Das komplette Dokument findet sich im Anhang.
Ausserdem wurde am Ende der Elaboration-Phase ein Code-Review geplant, an welchem der Code mit den Betreuer besprochen wird.

### 23.6.4. Tests

#### Unit Testing

Damit die Funktionalität des Programms überprüft werden kann, werden während dem Entwickeln der Software vor zu Test-Units erstellt. Diese Tests werden jeweils vor der Implementierung einer weiteren Programmfunktion erstellt und anschliessend durchgeführt.

#### Usability Tests

Es wird ein potentieller Benutzer ausgewählt, welcher die Software für die Gebrauchstauglichkeit überprüft. Während diese Person spezifische Aufgaben mit der Software erledigt, wird sie dabei beobachtet, um Schwierigkeiten bei der Verwendung der Applikation zu finden.

#### Systemtests

Systemtests enthalten folgende Ausführungen:

- Stimmt der Login-Prozess?
- Werden die Policy-Richtlinien richtig angewendet?
- Sind alle Use Cases richtig implementiert?

### 23.6.5. Versionskontrolle

Für das Projekt wird ein Git-Repository auf Github erstellt, auf welchem der gesamte Code, sowie alle Konfigurationsfiles abgelegt und versioniert werden. Es ist jederzeit ein lauffähiger und aktueller Release bereitgestellt, da zur Entwicklung von neuen Features auf einem separaten Branch gearbeitet wird.
Dokumentversionen werden ebenfalls mit Git auf Github versioniert, sind jedoch in einem anderen Repository untergebracht, damit der Code für eine spätere strongSwan-Implementierung und die Dokumentation dieser Bachelorarbeit sauber getrennt sind.

# 24. Iterationen und Meilensteine

## 24.1. Iterationsplanung

Diese Bachelorarbeit wird nach einem angepassten Rational Unified Process (RUP) geführt. Dieser definiert vier Phasen, welche durch Iterationen mehrfach durchlaufen werden.

- **Inception**
  Ziel der Inception ist die Definition der Aufgabenstellung des Projektes, sowie die Erstellung der Grunddokumente, welche während der Durchführung benötigt werden.
- **Elaboration 1**
  Die erste Elaboration-Iteration enthält die Planung des Projektes, die Erfassung der Anforderungen und die Erarbeitung der Analyse & Architektur.
- **Elaboration 2**
  Nach dieser Iteration sollte ein Prototyp (Alpha) stehen, in welchem alle Risiken abgeklärt sind.
- **Construction 1**
  Ist die 1. Construction beendet, sollte der Prototyp soweit erweitert sein, dass die MUSS-Kriterien gemäss Requirements-Dokument erfüllt sind (Beta).
- **Construction 2**
  Bei Bedarf Reserve-Zeit für die C1. Anhand des Projektstatus wird entschieden, welche weiteren Features realisiert werden. Nach der Iteration soll ein Release-Candidate bereitstehen.
- **Transition**
  Nach der Transition-Iteration sollten allfällige Fehler beseitigt und der Code stabilisiert sein. Ausserdem ist die Dokumentation der Arbeit fertiggestellt.

## 24.2. Meilensteine

Die folgenden Meilensteine wurde für das Projekt definiert:

| MS | Titel | Resultat | Datum |
|---|---|---|---|
| MS1 | Projektplan | Projektplan erstellt und alle Iterationen inkl. Daten geplant | SW04 |
| MS2 | Anforderungen und Analyse | Abschluss der Anforderungsspezifikation und Domainanalyse | SW06 |
| MS3 | Architektur/Design | Abschluss der Architektur- und Designentscheide | SW08 |
| MS4 | End-Of-Elaboration | Prototyp festgelegt, Release 0.1a Alpha, welcher die Risiken beseitigt | SW10 |
| MS5 | End-Of-Construction 1 | Release 0.1b Beta, welcher die wichtigsten Features realisiert. | SW14 |
| MS6 | Abgabe | Release 1.0 | SW17 |

Tabelle 24.1.: Meilensteine

## 24.3. Zeitplan

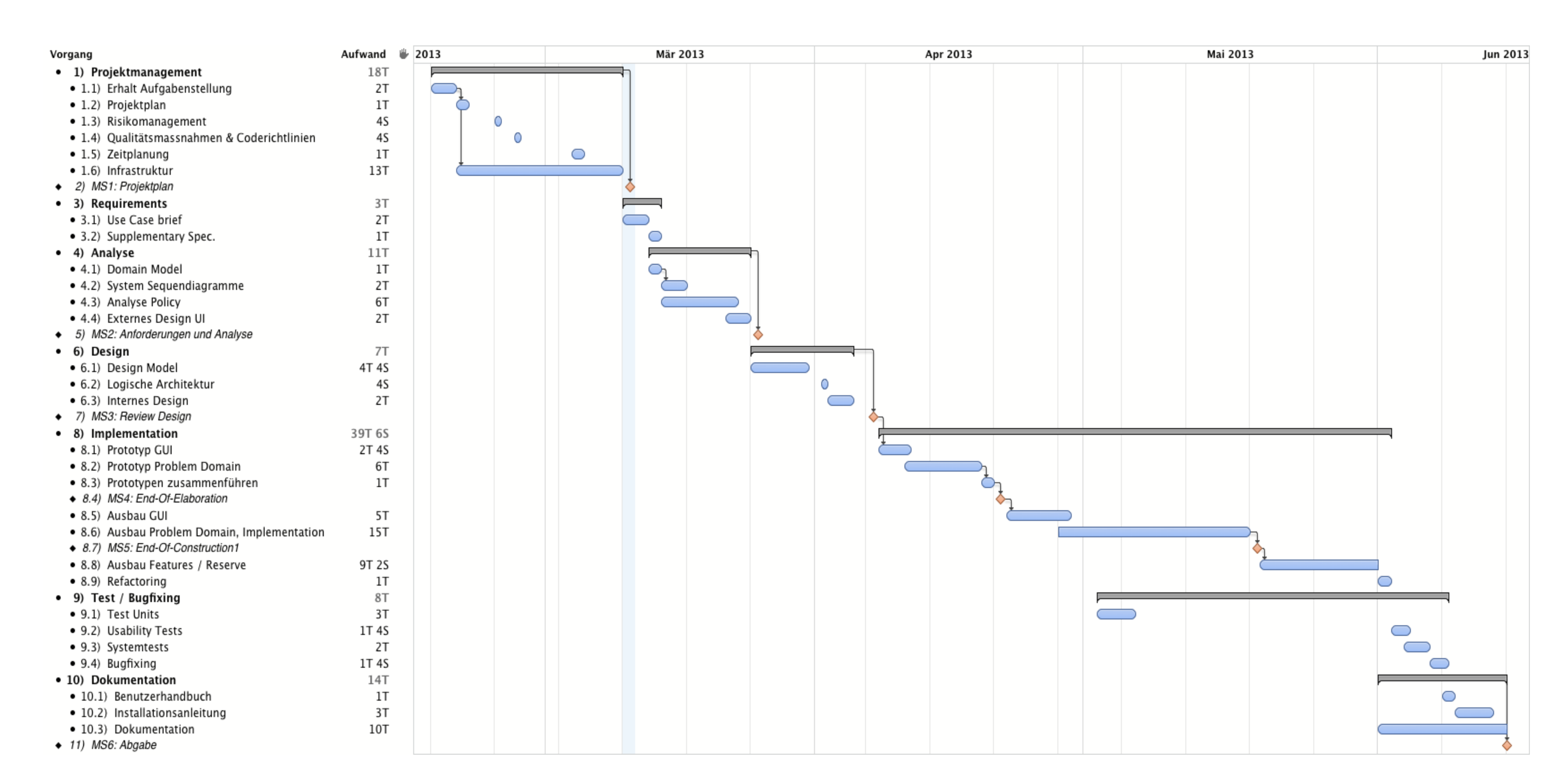



Abbildung 24.1.: Gantt-Chart

## 24.4. Arbeitspakete

| Nr. | Name | Inhalt | Iteration | Prio |
|---|---|---|---|---|
| **0x** | **Projektmanagement** | | | |
| 00 | Erhalt Aufgabenstellung | Kick-off und Einarbeitung | Inception | 1 |
| 01 | Projektplan | Projektplan wird erstellt | Elaboration 1 | 1 |
| 02 | Risikomanagement | Risikoanalyse und Massnahmen definieren | Elaboration 1 | 3 |
| 03 | Qualitätsmassnahmen & Coderichtlinien | Definieren von Qualitätsmassnahmen und Erarbeiten von Codierungsrichtlinen | Elaboration 1 | 1 |
| 04 | Zeitplanung | Arbeitszeiteneinteilung | Elaboration 1 | 1 |
| 05 | Infrastruktur | Einrichten der Infrastruktur | Elaboration 1 | 2 |
| 06 | MS1: Review Projektplan | Review und Korrektur Projektplan | Elaboration 1 | 1 |
| **1x** | **Requirements** | | | |
| 10 | Use Case brief | Erarbeiten aller Use Cases | Elaboration 1 | 2 |
| 11 | Supplementary Spec. | Erfassen der nicht-funktionalen Anforderungen | Elaboration 1 | 2 |
| **2x** | **Analyse** | | | |
| 20 | Domain Model | Domain Model aus Use Cases ausarbeiten | Elaboration 1 | 1 |
| 21 | System Sequenzdiagramme | Darstellen der wichtigsten internen Abläufe | Elaboration 1 | 1 |
| 22 | Analyse Policy | Ausarbeiten der BYOD Policy Rule Datenbankschnittstelle | Elaboration 1 | 1 |
| 23 | Externes Design UI | Erstellen der Paper-Prototypes | Elaboration 1 | 2 |
| 24 | MS2: Review Requirements/Analyse | Review und Korrektur Requirements/Analyse | Elaboration 1 | 1 |
| **3x** | **Design** | | | |
| 30 | Design Model | Erstellen des Klassendiagramms | Elaboration 2 | 2 |
| 31 | Logische Architektur | Festlegen der logischen Struktur | Elaboration 2 | 1 |
| 32 | Internes Design | Internes Design festlegen | Elaboration 2 | 2 |
| 33 | MS3: Review Design | Review und Korrektur Architektur/Design | Elaboration 2 | 2 |

Tabelle 24.2.: Arbeitspakete Teil 1

| **4x** | **Implementation** | | | |
|---|---|---|---|---|
| 40 | Prototyp GUI | Prototyp des GUIs erstellen | Elaboration 2 | 2 |
| 41 | Prototyp Problem Domain | Erster Prototyp | Elaboration 2 | 2 |
| 42 | Prototypen zusammenführen | Erarbeiten des Release 0.1 | Elaboration 2 | 2 |
| 43 | MS4: End-Of-Elaboration | Review Prototyp (Alpha) | Elaboration 2 | 1 |
| 44 | Ausbau GUI | GUI auf Endzustand ausbauen | Construction 1 | 2 |
| 45 | Ausbau Problem Domain, Implementation | Erweitern des Codes um die wichtigsten Usecases | Construction 1 | 2 |
| 46 | MS5: End-Of-Construction 1 | Review Prototyp (Beta) | Construction 1 | 1 |
| 47 | Ausbau Features / Reserve | Erweitern des Codes um alle Features, inkl. optionale | Construction 2 | 3 |
| 48 | Refactoring | Codeoptimierungen durchführen | Construction 2 | 2 |
| **5x** | **Test / Bugfixing** | | | |
| 50 | Test Units | Test Cases für zwingende Funktionen erarbeiten und ausführen | Elaboration 2 | 2 |
| 51 | Usability Tests | Usability Tests definieren und durchführen | Construction 2 | 3 |
| 52 | Systemtests | Funktionale Systemtests definieren und durchführen | Construction 2 | 2 |
| 53 | Bugfixing | Gefundene Fehler beheben | Construction 2 | 2 |
| **6x** | **Dokumentation** | | | |
| 60 | Benutzerhandbuch | Benutzerhandbuch erstellen | Transition | 3 |
| 61 | Installationsanleitung | Installationsanleitung erstellen | Transition | 2 |
| 62 | Dokumentation | BA-Dokumentation erstellen | Transition | 2 |
| 63 | MS6: Abgabe | Abgabe Release Candidate und Dokumentation | Transition | 1 |
| 64 | BA-Schlusspräsentation | Schlusspräsentation vorbereiten und halten | Transition | 2 |

Tabelle 24.3.: Arbeitspakete Teil 2

# 25. Risikomanagement

## 25.1. Projektspezifische Risiken

| **R1: Policy Rule DB-Schnittstelle ist nicht praktikabel** | |
|---|---|
| **Beschreibung** | Ausarbeitung, welche Gesundheitstests beim Login in ein Firmennetz auf der Basis des früheren Verhaltens jeweils von einem spezifischen Android Gerät verlangt werden sollen. |
| **Massnahmen** | Genaue Analyse der verschiedenen Login-Prozessen und Verhalten der Android-Pakete. Nach Erarbeiten des Konzeptes Review mit Betreuer |
| **Vorgehen bei Eintreffen** | Konzept überdenken und verbessern |

Tabelle 25.1.: Risikotabelle 1

| **R2: Probleme mit Python/Django-Framework** | |
|---|---|
| **Beschreibung** | Die zum Teil für die Mitarbeiter neuen Technologien benötigen mehr Zeit für die Einarbeitung oder unterstützen möglicherweise geforderte Funktionen nicht |
| **Massnahmen** | Einarbeitung in Internet-Technologien, Finden von Work-Arounds, Zeitreserve einplanen |
| **Vorgehen bei Eintreffen** | Mehraufwand |

Tabelle 25.2.: Risikotabelle 2

| **R3: Probleme mit der Datenbank** | |
|---|---|
| **Beschreibung** | Das gewählte Datenbanksystem SQLite unterstützt nicht alle geforderten Funktionen oder das entworfene Schema hält Anforderungen nicht aus |
| **Massnahmen** | Frühes Modellieren eines geeigneten Datenbankschemas und Erstellen eines Prototyps, um zu testen, ob die Anforderungen abgedeckt werden können |
| **Vorgehen bei Eintreffen** | Mehraufwand bei Schemaänderung oder Datenbank-Migration auf z.B. MySQL |

Tabelle 25.3.: Risikotabelle 3

| R4: Kommunikation Django ↔ strongSwan IMV | |
|---|---|
| **Beschreibung** | Kommunikation und Interaktion des Django-Frameworks mit strongSwan IMV kann nicht korrekt ausgeführt werden. |
| **Massnahmen** | Proof of Technology frühzeitig erarbeiten |
| **Vorgehen bei Eintreffen** | Mehraufwand |

Tabelle 25.4.: Risikotabelle 4

## 25.2. Allgemeine Risiken

| R5: Ausfall der Infrastruktur | |
|---|---|
| **Beschreibung** | HW eines Projektmitgliedes fällt aus, Github-Server fällt aus, Netzwerk-Infrastruktur der HSR fällt aus |
| **Massnahmen** | 1. Alternative HW organisieren<br>2. Kopien auf Arbeitsrechner aktuell halten<br>3. mehrere Möglichkeiten für Datenaustausch innerhalb des Teams bereithalten |
| **Vorgehen bei Eintreffen** | an Arbeitsplatzrechner in HSR BA-Zimmer arbeiten, Neuer Git Server aufsetzen, Daten über alternatives Netzwerk/Medien austauschen |

Tabelle 25.5.: Risikotabelle 5

| R6: Datenverlust | |
|---|---|
| **Beschreibung** | erarbeitete Projekt-Artefakte werden lokal unwiderruflich geändert oder gelöscht |
| **Massnahmen** | Artefakte konsequent unter Versionsverwaltung (Github) stellen, lokale Kopien |
| **Vorgehen bei Eintreffen** | Dateien aus letztem Backup vom letzten Build wiederherstellen und ev. auf den neuesten Stand bringen |

Tabelle 25.6.: Risikotabelle 6

| R7: Fehleinschätzung des Aufwandes | |
|---|---|
| **Beschreibung** | Zeitplan wird nicht eingehalten, da der Aufwand von einzelnen Arbeitspaketen falsch eingeschätzt wurde |
| **Massnahmen** | kritische Arbeitspakete (Domainmodell, Backup-Planung, Konfigurationsverwaltung) frühzeitig (Elaboration 1 und 2) bearbeiten |
| **Vorgehen bei Eintreffen** | nicht alle geplanten Funktionen und Features umsetzen, Mehrarbeit |

Tabelle 25.7.: Risikotabelle 7

| **R8: Ausfall eines Projektmitglieds** | |
|---|---|
| **Beschreibung** | Ausfall eines Projektmitgliedes aufgrund Krankheit, Unfall, Studiumabbruch |
| **Massnahmen** | Verantwortlichkeiten in Team klar definieren |
| **Vorgehen bei Eintreffen** | Arbeit innerhalb des bestehenden Teams aufteilen, Funktionsumfang kürzen |

Tabelle 25.8.: Risikotabelle 8

## 25.3. Risikoschätzung

| **Nr** | **Titel** | **max. Schaden [h]** | **Eintritts-Wsk** | **gew. Schaden [h]** |
|---|---|---|---|---|
| **Projektspezifische Risiken** | | | | |
| R1 | Policy Rule DB-Schnittstelle ist nicht praktikabel | 40 | 20.00 % | 8.00 |
| R2 | Probleme mit dem Django-Framework | 80 | 10.00 % | 8.00 |
| R3 | Probleme mit der Datenbank | 80 | 5.00 % | 4.00 |
| R4 | Kommunikation Django $\leftrightarrow$ strongSwan IMV | 40 | 2.00 % | 0.80 |
| **Allgemeine Risiken** | | | | |
| R5 | Ausfall der Infrastruktur | 40 | 0.50% | 0.20 |
| R6 | Datenverlust | 600 | 0.50 % | 3.00 |
| R7 | Fehleinschätzung des Aufwandes | 60 | 20.00 % | 12.00 |
| R8 | Ausfall eines Projektmitglieds | 370 | 0.50 % | 1.85 |
| **Summe** | | **1310** | **58.50 %** | **39.7** |

Tabelle 25.9.: Risikoschätzung

# 26. Persönliche Berichte

## 26.1. Stefan Rohner

In den letzten 14 Wochen bearbeitete ich mit meinem Kollegen Marco Tanner eine sehr spannende aber zum Teil konzeptuell auch anspruchsvolle Arbeit. Wir waren fast immer gleicher Meinung und sehr motiviert, ein funktionsfähiges Produkt zu entwickeln. Sehr angenehm war auch die Zusammenarbeit mit unseren Betreuern Prof. Dr. Steffen und Tobias Brunner von ITA, welche wichtige konzeptuelle Inputs geben konnten.

Zu den für mich interessantesten Punkten gehörten unter anderem:

- Erkennen der Zusammenhängen der TNC Architektur
- Kennenlernen des Python-Framework Django
- Erstellen des GUI mit modernen Web-Frameworks wie JQuery und Twitter's Bootstrap

Eine spannende Erfahrung war ausserdem, dass die gesamte Entwicklung auf OpenSource-Software basierte, was ich aus meinem geschäftlichen Umfeld weniger kenne. Der Projektablauf lief meines Erachtens sehr reibungsfrei ab. Den auf unser Projekt abgestimmten RUP erwies sich als sehr hilfreich und auch das Projektmanagement konnten wir gut bewältigen. Obwohl ich das Gefühl hatte, dass wir die grundlegende Architektur schon sehr früh sehr realitätsnah definiert hatten, konnten ich feststellen, dass man erst beim Implementieren die Dinge aus einer anderen Sicht betrachtet und somit auf eine andere Weise gelöst hätte. Somit mussten wir noch bis am Schluss der Implementationsphase entscheiden, ob wir gewisse Dinge noch erweitern bzw. anpassen konnten, oder diese ganz bleiben lassen mussten.
Ich finde aber wir haben eine sehr ansprechende Applikation entwickeln können, welche wir so gerne für den Einsatz oder zur weiteren Entwicklung übergeben können.
Abschliessend kann ich behaupten, sehr viel dazugelernt zu haben, vor allem im Bereich der Anwendung von Webtechnologien.

## 26.2. Marco Tanner

Die Ausschreibung der Bachelorarbeit “BYOD Malware Database Management Tool for Android” hatte mich seit Bekanntgabe der Arbeiten angesprochen, vor allem wegen der Freiheit, eine beliebige technologische Grundlage für das Projekt wählen zu dürfen, solange es freie Software war. Zu dem Zeitpunkt hatte ich aber geplant, die Arbeit als Einzelprojekt in Angriff zu nehmen, da sich mein SA-Team aufgelöst hatte.
Auf die Email von Stefan an alle Einzelgänger hatten wir uns zusammengesetzt und entschieden, ein Team zu bilden.
Obwohl es ein gewisses Risiko darstellt, mit jemandem zusammenzuarbeiten den man bisher noch nicht kennt, war es auf jeden Fall die richtige Entscheidung, unsere Fähigkeiten haben sich gut ergänzt und die Arbeitsteilung und Planung lief reibungsfrei. Während sich Stefan hauptsächlich auf das Frontend unserer Applikation in Sachen Konzept, Design und Umsetzung konzentrierte, durfte ich mich dem Backend, also dem Datenmodell, der Schnittstelle zur Datenbank und zu strongSwan und der Erstellung von Views in Python widmen.
Während der Arbeit konnte ich viel über den modernen, eleganten Ansatz lernen, den Django verfolgt um Webseiten schnell und nach dem DRY-Prinzip zu erstellen. Ein Framework ohne vorherige Erfahrungen zu verwenden birgt immer gewisse Hürden. Einige Male während des Projektverlaufs wurden wir per Zufall auf Django-Features aufmerksam, die uns wohl die Arbeit erleichtert hätten, aber nicht mehr umgesetzt werden konnten, da zu viel Code geändert hätte werden müssen.

Als Ganzes gesehen bin ich mit unserer Arbeit zufrieden und erachte es als würdigen Abschluss für mein Studium an der HSR. Ich bin überzeugt davon, dass mir die gesammelten Erfahrungen bei zukünftigen Projekten von Nutzen sein werden.

# Teil VII.

# Appendix

# Inhaltsverzeichnis

# A. Glossar

- **Client**
  Unter Client wird ein Android-Gerät verstanden, das sich in ein strongSwan VPN einwählen will und von den definierten Cygnet-Policies betroffen ist.
- **Cygnet**
  Das in dieser Arbeit geplante und entwickelte Produkt. Es umfasst eine Web-Applikation zur Verwaltung von Policies und eine Schnittstelle zu den strongSwan IMVs.
- **Benutzer**
  Der Benutzer von Cygnet. Gemäss Anforderungen ein System-Administrator oder Ähnliches.
- **Enforcement**
  Erzwingt eine Policy auf einer Gruppe von Clients.
- **IMC**
  Information Measurement Collector. Sammelt Daten auf dem Client und sendet diese an den IMV zur Verifizierung. Erzwingt Policies auf Clientseite.
- **IMV**
  Information Measurement Verifier. Empfängt Informationen vom IMC und verifiziert diese. Erzwingt Policies auf Serverseite.
- **Package**
  Ein Software-Paket, das auf einem Client installiert ist. Kann auf eine schwarze Liste gesetzt werden.
- **Policy**
  Eine Richtlinie, die ein Client einzuhalten hat, wenn er sich ins VPN einwählen will.
- **Product**
  Ein Produkt, bzw. Betriebssystem, das auf einem Client installiert ist.
- **strongSwan**
  Eine in C geschriebene Open-Source IPSec Implementierung für Linux und Android.
- **TNC(-Server)**
  Trusted Network Connect(-Server). Eine Architektur der Trusted Computing Group für Network Access Control. Der dazugehörige Server ist im Falle von strongSwan die Verwaltungsinstanz der IMVs.

- **VPN**
  Virtual Private Network. Eine verschlüsselte Verbindung, die dem Benutzer Zugang ins lokale Netzwerk einer Firma oder Organisation über einen unsicheren Kanal (z.B. das Internet) ermöglicht.
- **Workitem**
  Definiert einen Arbeitsauftrag. Wird von Cygnet für IMVs generiert und von diesen ausgeführt. Die Resultate werden zurück an Cygnet geliefert und dann von Cygnet ausgewertet.

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Listings

# B. Listing Datenmodell

```
CREATE TABLE "products" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" text NOT NULL
);
CREATE TABLE "devices" (
    "id" integer NOT NULL PRIMARY KEY,
    "value" text NOT NULL,
    "description" text,
    "product_id" integer NOT NULL REFERENCES "products" ("id"),
    "created" datetime
);
CREATE TABLE "groups_members" (
    "id" integer NOT NULL PRIMARY KEY,
    "group_id" integer NOT NULL,
    "device_id" integer NOT NULL REFERENCES "devices" ("id"),
    UNIQUE ("group_id", "device_id")
);
CREATE TABLE "groups_product_defaults" (
    "id" integer NOT NULL PRIMARY KEY,
    "group_id" integer NOT NULL,
    "product_id" integer NOT NULL REFERENCES "products" ("id"),
    UNIQUE ("group_id", "product_id")
);
CREATE TABLE "groups" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(50) NOT NULL,
    "parent" integer
);
CREATE TABLE "directories" (
    "id" integer NOT NULL PRIMARY KEY,
    "path" text NOT NULL UNIQUE
);
CREATE TABLE "files" (
    "id" integer NOT NULL PRIMARY KEY,
    "dir" integer NOT NULL REFERENCES "directories" ("id"),
    "name" text NOT NULL
);
CREATE TABLE "algorithms" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(20) NOT NULL
);
CREATE TABLE "file_hashes" (
    "id" integer NOT NULL PRIMARY KEY,
    "file" integer NOT NULL REFERENCES "files" ("id"),
    "product" integer NOT NULL REFERENCES "products" ("id"),
    "key" integer NOT NULL,
    "algo" integer NOT NULL REFERENCES "algorithms" ("id"),
    "hash" blob NOT NULL
);
CREATE TABLE "packages" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" text NOT NULL UNIQUE,
    "blacklist" integer NOT NULL
);
CREATE TABLE "versions" (
    "id" integer NOT NULL PRIMARY KEY,
    "package" integer NOT NULL REFERENCES "packages" ("id"),
    "product" integer NOT NULL REFERENCES "products" ("id"),
    "release" text NOT NULL,
    "security" bool NOT NULL,
```

```
61     "time" datetime NOT NULL,
62     "blacklist" integer
63 );
64 CREATE TABLE "policies" (
65     "id" integer NOT NULL PRIMARY KEY,
66     "type" integer NOT NULL,
67     "name" varchar(100) NOT NULL UNIQUE,
68     "argument" text,
69     "rec_fail" integer NOT NULL,
70     "rec_noresult" integer NOT NULL,
71     "file" integer REFERENCES "files" ("id"),
72     "dir" integer REFERENCES "directories" ("id")
73 );
74 CREATE TABLE "enforcements" (
75     "id" integer NOT NULL PRIMARY KEY,
76     "policy" integer NOT NULL REFERENCES "policies" ("id"),
77     "group_id" integer NOT NULL REFERENCES "groups" ("id"),
78     "max_age" integer NOT NULL,
79     "rec_fail" integer,
80     "rec_noresult" integer,
81     UNIQUE ("policy", "group_id")
82 );
83 CREATE TABLE "identities" (
84     "id" integer NOT NULL PRIMARY KEY,
85     "value" text NOT NULL
86 );
87 CREATE TABLE "sessions" (
88     "id" integer NOT NULL PRIMARY KEY,
89     "connection" integer NOT NULL,
90     "device" integer NOT NULL REFERENCES "devices" ("id"),
91     "identity" integer NOT NULL REFERENCES "identities" ("id"),
92     "time" datetime NOT NULL,
93     "rec" integer
94 );
95 CREATE TABLE "workitems" (
96     "id" integer NOT NULL PRIMARY KEY,
97     "enforcement" integer NOT NULL REFERENCES "enforcements" ("id"),
98     "session" integer NOT NULL REFERENCES "sessions" ("id"),
99     "type" integer NOT NULL,
100    "argument" text NOT NULL,
101    "fail" integer,
102    "noresult" integer,
103    "result" text,
104    "recommendation" integer,
105    "file_id" integer REFERENCES "files" ("id"),
106    "dir_id" integer REFERENCES "directories" ("id")
107 );
108 CREATE TABLE "results" (
109    "id" integer NOT NULL PRIMARY KEY,
110    "session" integer NOT NULL REFERENCES "sessions" ("id"),
111    "policy" integer NOT NULL REFERENCES "policies" ("id"),
112    "result" text NOT NULL,
113    "rec" integer NOT NULL
114 );
```

../architecture/models.sql

**PEP:** 8
**Title:** Style Guide for Python Code
**Version:** 68852
**Last-Modified:** 2009-01-22 09:36:39 +0100 (Thu, 22 Jan 2009)
**Author:** Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>
**Status:** Active
**Type:** Process
**Created:** 05-Jul-2001
**Post-History:** 05-Jul-2001

# Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution.  Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python[1].

This document was adapted from Guido's original Python Style Guide essay[2], with some additions from Barry's style guide[5].  Where there's conflict, Guido's style rules for the purposes of this PEP.  This PEP may still be incomplete (in fact, it may never be finished <wink>).

# A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written.  The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.  As PEP 20 [6] says, "Readability counts".

A style guide is about consistency.  Consistency with this style guide is important.  Consistency within a project is more important. Consistency within one module or function is most important.

But most importantly: know when to be inconsistent -- sometimes the style guide just doesn't apply.  When in doubt, use your best judgment.  Look at other examples and decide what looks best.  And don't hesitate to ask!

Two good reasons to break a particular rule:

(1) When applying the rule would make the code less readable, even for someone who is used to reading code that follows the rules.

(2) To be consistent with surrounding code that also breaks it (maybe for historic reasons) -- although this is also an opportunity to clean up someone else's mess (in true XP style).

## *Code lay-out*

Indentation

Use 4 spaces per indentation level.

For really old code that you don't want to mess up, you can continue to use 8-space tabs.

Tabs or Spaces?

Never mix tabs and spaces.

The most popular way of indenting Python is with spaces only. The second-most popular way is with tabs only. Code indented with a mixture of tabs and spaces should be converted to using spaces exclusively. When invoking the Python command line interpreter with the -t option, it issues warnings about code that illegally mixes tabs and spaces. When using -tt these warnings become errors. These options are highly recommended!

For new projects, spaces-only are strongly recommended over tabs. Most editors have features that make this easy to do.

Maximum Line Length

Limit all lines to a maximum of 79 characters.

There are still many devices around that are limited to 80 character lines; plus, limiting windows to 80 characters makes it possible to have several windows side-by-side. The default wrapping on such devices disrupts the visual structure of the code, making it more difficult to understand. Therefore, please limit all lines to a maximum of 79 characters. For flowing long blocks of text (docstrings or comments), limiting the length to 72 characters is recommended.

The preferred way of wrapping long lines is by using Python's implied line continuation inside parentheses, brackets and braces. If necessary, you

can add an extra pair of parentheses around an expression, but sometimes using a backslash looks better. Make sure to indent the continued line appropriately. The preferred place to break around a binary operator is *after* the operator, not before it. Some examples:

```
class Rectangle(Blob):

    def __init__(self, width, height,
                 color='black', emphasis=None, highlight=0):
        if width == 0 and height == 0 and \
           color == 'red' and emphasis == 'strong' or \
           highlight > 100:
            raise ValueError("sorry, you lose")
        if width == 0 and height == 0 and (color == 'red' or
                                           emphasis is None):
            raise ValueError("I don't think so -- values are %s, %s" %
                             (width, height))
        Blob.__init__(self, width, height,
                      color, emphasis, highlight)
```

## Blank Lines

Separate top-level function and class definitions with two blank lines.

Method definitions inside a class are separated by a single blank line.

Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).

Use blank lines in functions, sparingly, to indicate logical sections.

Python accepts the control-L (i.e. ^L) form feed character as whitespace; Many tools treat these characters as page separators, so you may use them to separate pages of related sections of your file.

## Encodings (PEP 263)

Code in the core Python distribution should aways use the ASCII or Latin-1 encoding (a.k.a. ISO-8859-1). For Python 3.0 and beyond, UTF-8 is preferred over Latin-1, see PEP 3120.

Files using ASCII (or UTF-8, for Python 3.0) should not have a coding cookie. Latin-1 (or UTF-8) should only be used when a comment or docstring needs to mention an author name that requires Latin-1; otherwise, using \x, \u or \U escapes is the preferred

way to include non-ASCII data in string literals.

For Python 3.0 and beyond, the following policy is prescribed for the standard library (see PEP 3131): All identifiers in the Python standard library MUST use ASCII-only identifiers, and SHOULD use English words wherever feasible (in many cases, abbreviations and technical terms are used which aren't English). In addition, string literals and comments must also be in ASCII. The only exceptions are (a) test cases testing the non-ASCII features, and (b) names of authors. Authors whose names are not based on the latin alphabet MUST provide a latin transliteration of their names.

Open source projects with a global audience are encouraged to adopt a similar policy.

## *Imports*

- Imports should usually be on separate lines, e.g.:

```
Yes: import os
     import sys

No:  import sys, os
```

it's okay to say this though:

```
from subprocess import Popen, PIPE
```

- Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

  Imports should be grouped in the following order:

  1. standard library imports
  2. related third party imports
  3. local application/library specific imports

  You should put a blank line between each group of imports.

  Put any relevant __all__ specification after the imports.

- Relative imports for intra-package imports are highly discouraged.

Always use the absolute package path for all imports.
Even now that PEP 328 [7] is fully implemented in Python 2.5,
its style of explicit relative imports is actively discouraged;
absolute imports are more portable and usually more readable.

- When importing a class from a class-containing module, it's usually okay
  to spell this

```
from myclass import MyClass
from foo.bar.yourclass import YourClass
```

If this spelling causes local name clashes, then spell them

```
import myclass
import foo.bar.yourclass
```

and use "myclass.MyClass" and "foo.bar.yourclass.YourClass"

## *Whitespace in Expressions and Statements*

Pet Peeves

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses, brackets or braces.

Yes: `spam(ham[1], {eggs: 2})`
No: `spam( ham[ 1 ], { eggs: 2 } )`

- Immediately before a comma, semicolon, or colon:

Yes: `if x == 4: print x, y; x, y = y, x`
No: `if x == 4 : print x , y ; x , y = y , x`

- Immediately before the open parenthesis that starts the argument
  list of a function call:

Yes: `spam(1)`
No: `spam (1)`

- Immediately before the open parenthesis that starts an indexing or
  slicing:

Yes: `dict['key'] = list[index]`

No: `dict ['key'] = list [index]`

- More than one space around an assignment (or other) operator to align it with another.

  Yes:

```
x = 1
y = 2
long_variable = 3
```

  No:

```
x             = 1
y             = 2
long_variable = 3
```

Other Recommendations

- Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -= etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

- Use spaces around arithmetic operators:

  Yes:

```
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

  No:

```
i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

- Don't use spaces around the '=' sign when used to indicate a keyword argument or a default parameter value.

  Yes:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

No:

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

- Compound statements (multiple statements on the same line) are generally discouraged.

  Yes:

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

  Rather not:

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

- While sometimes it's okay to put an if/for/while with a small body on the same line, never do this for multi-clause statements.  Also avoid folding such long lines!

  Rather not:

```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

  Definitely not:

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()

try: something()
finally: cleanup()

do_one(); do_two(); do_three(long, argument,
                             list, like, this)

if foo == 'blah': one(); two(); three()
```

## *Comments*

Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!

Comments should be complete sentences. If a comment is a phrase or sentence, its first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!).

If a comment is short, the period at the end can be omitted. Block comments generally consist of one or more paragraphs built out of complete sentences, and each sentence should end in a period.

You should use two spaces after a sentence-ending period.

When writing English, Strunk and White apply.

Python coders from non-English speaking countries: please write your comments in English, unless you are 120% sure that the code will never be read by people who don't speak your language.

### Block Comments

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (unless it is indented text inside the comment).

Paragraphs inside a block comment are separated by a line containing a single #.

### Inline Comments

Use inline comments sparingly.

An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with a # and a single space.

Inline comments are unnecessary and in fact distracting if they state the obvious. Don't do this:

```
x = x + 1                 # Increment x
```

But sometimes, this is useful:

```
x = x + 1                 # Compensate for border
```

Documentation Strings

Conventions for writing good documentation strings (a.k.a. "docstrings") are immortalized in PEP 257 [3].

- Write docstrings for all public modules, functions, classes, and methods. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the "def" line.

- PEP 257 describes good docstring conventions. Note that most importantly, the """ that ends a multiline docstring should be on a line by itself, and preferably preceded by a blank line, e.g.:

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.

"""
```

- For one liner docstrings, it's okay to keep the closing """ on the same line.

## *Version Bookkeeping*

If you have to have Subversion, CVS, or RCS crud in your source file, do it as follows.

```
__version__ = "$Revision: 68852 $"
# $Source$
```

These lines should be included after the module's docstring, before any other code, separated by a blank line above and below.

## Naming Conventions

The naming conventions of Python's library are a bit of a mess, so we'll never get this completely consistent -- nevertheless, here are the currently recommended naming standards. New modules and packages (including third party frameworks) should be written to these standards, but where an existing library has a different style, internal consistency is preferred.

Descriptive: Naming Styles

There are a lot of different naming styles. It helps to be able to recognize what naming style is being used, independently from what they are used for.

The following naming styles are commonly distinguished:

- b (single lowercase letter)

- B (single uppercase letter)

- lowercase

- lower_case_with_underscores

- UPPERCASE

- UPPER_CASE_WITH_UNDERSCORES

- CapitalizedWords (or CapWords, or CamelCase -- so named because of the bumpy look of its letters[4]). This is also sometimes known as StudlyCaps.

  Note: When using abbreviations in CapWords, capitalize all the letters of the abbreviation. Thus HTTPServerError is better than HttpServerError.

- mixedCase (differs from CapitalizedWords by initial lowercase character!)

- Capitalized_Words_With_Underscores (ugly!)

There's also the style of using a short unique prefix to group related names together. This is not used much in Python, but it is mentioned for completeness. For example, the os.stat() function returns a tuple whose

items traditionally have names like st_mode, st_size, st_mtime and so on. (This is done to emphasize the correspondence with the fields of the POSIX system call struct, which helps programmers familiar with that.)

The X11 library uses a leading X for all its public functions. In Python, this style is generally deemed unnecessary because attribute and method names are prefixed with an object, and function names are prefixed with a module name.

In addition, the following special forms using leading or trailing underscores are recognized (these can generally be combined with any case convention):

- _single_leading_underscore: weak "internal use" indicator. E.g. "from M import *" does not import objects whose name starts with an underscore.

- single_trailing_underscore_: used by convention to avoid conflicts with Python keyword, e.g.

  ```
  Tkinter.Toplevel(master, class_='ClassName')
  ```

- __double_leading_underscore: when naming a class attribute, invokes name mangling (inside class FooBar, __boo becomes _FooBar__boo; see below).

- __double_leading_and_trailing_underscore__: "magic" objects or attributes that live in user-controlled namespaces. E.g. __init__, __import__ or __file__. Never invent such names; only use them as documented.

## Prescriptive: Naming Conventions

### Names to Avoid

Never use the characters `l' (lowercase letter el), `O' (uppercase letter oh), or `I' (uppercase letter eye) as single character variable names.

In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use `l', use `L' instead.

### Package and Module Names

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.

Since module names are mapped to file names, and some file systems are case insensitive and truncate long names, it is important that module names be chosen to be fairly short -- this won't be a problem on Unix, but it may be a problem when the code is transported to older Mac or Windows versions, or DOS.

When an extension module written in C or C++ has an accompanying Python module that provides a higher level (e.g. more object oriented) interface, the C/C++ module has a leading underscore (e.g. _socket).

Class Names

Almost without exception, class names use the CapWords convention. Classes for internal use have a leading underscore in addition.

Exception Names

Because exceptions should be classes, the class naming convention applies here. However, you should use the suffix "Error" on your exception names (if the exception actually is an error).

Global Variable Names

(Let's hope that these variables are meant for use inside one module only.) The conventions are about the same as those for functions.

Modules that are designed for use via "from M import *" should use the __all__ mechanism to prevent exporting globals, or use the older convention of prefixing such globals with an underscore (which you might want to do to indicate these globals are "module non-public").

Function Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

Function and method arguments

Always use 'self' for the first argument to instance methods.

Always use 'cls' for the first argument to class methods.

If a function argument's name clashes with a reserved keyword, it is generally better to append a single trailing underscore rather than use an abbreviation or spelling corruption. Thus "print_" is better than "prnt". (Perhaps better is to avoid such clashes by using a synonym.)

### Method Names and Instance Variables

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

Use one leading underscore only for non-public methods and instance variables.

To avoid name clashes with subclasses, use two leading underscores to invoke Python's name mangling rules.

Python mangles these names with the class name: if class Foo has an attribute named __a, it cannot be accessed by Foo.__a. (An insistent user could still gain access by calling Foo._Foo__a.) Generally, double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

Note: there is some controversy about the use of __names (see below).

### Constants

Constants are usually declared on a module level and written in all capital letters with underscores separating words. Examples include MAX_OVERFLOW and TOTAL.

### Designing for inheritance

Always decide whether a class's methods and instance variables (collectively: "attributes") should be public or non-public. If in doubt, choose non-public; it's easier to make it public later than to make a public attribute non-public.

Public attributes are those that you expect unrelated clients of your class to use, with your commitment to avoid backward incompatible changes. Non-public attributes are those that are not intended to be used by third parties; you make no guarantees that non-public attributes won't change or even be removed.

We don't use the term "private" here, since no attribute is really private in Python (without a generally unnecessary amount of work).

Another category of attributes are those that are part of the "subclass API" (often called "protected" in other languages). Some classes are designed to be inherited from, either to extend or modify aspects of the class's behavior. When designing such a class, take care to make explicit decisions about which attributes are public, which are part of the subclass API, and which are truly only to be used by your base class.

With this in mind, here are the Pythonic guidelines:

- Public attributes should have no leading underscores.

- If your public attribute name collides with a reserved keyword, append a single trailing underscore to your attribute name. This is preferable to an abbreviation or corrupted spelling. (However, notwithstanding this rule, 'cls' is the preferred spelling for any variable or argument which is known to be a class, especially the first argument to a class method.)

  Note 1: See the argument name recommendation above for class methods.

- For simple public data attributes, it is best to expose just the attribute name, without complicated accessor/mutator methods. Keep in mind that Python provides an easy path to future enhancement, should you find that a simple data attribute needs to grow functional behavior. In that case, use properties to hide functional implementation behind simple data attribute access syntax.

  Note 1: Properties only work on new-style classes.

  Note 2: Try to keep the functional behavior side-effect free, although side-effects such as caching are generally fine.

  Note 3: Avoid using properties for computationally expensive operations; the attribute notation makes the caller believe that access is (relatively) cheap.

- If your class is intended to be subclassed, and you have attributes that you do not want subclasses to use, consider naming them with double leading underscores and no trailing underscores. This invokes Python's name mangling algorithm, where the name of the class is mangled into the attribute name. This helps avoid attribute name collisions should subclasses inadvertently contain attributes with the same name.

  Note 1: Note that only the simple class name is used in the mangled

name, so if a subclass chooses both the same class name and attribute name, you can still get name collisions.

Note 2: Name mangling can make certain uses, such as debugging and __getattr__(), less convenient. However the name mangling algorithm is well documented and easy to perform manually.

Note 3: Not everyone likes name mangling. Try to balance the need to avoid accidental name clashes with potential use by advanced callers.

## Programming Recommendations

- Code should be written in a way that does not disadvantage other implementations of Python (PyPy, Jython, IronPython, Pyrex, Psyco, and such).

  For example, do not rely on CPython's efficient implementation of in-place string concatenation for statements in the form a+=b or a=a+b. Those statements run more slowly in Jython. In performance sensitive parts of the library, the ''.join() form should be used instead. This will ensure that concatenation occurs in linear time across various implementations.

- Comparisons to singletons like None should always be done with 'is' or 'is not', never the equality operators.

  Also, beware of writing "if x" when you really mean "if x is not None" -- e.g. when testing whether a variable or argument that defaults to None was set to some other value. The other value might have a type (such as a container) that could be false in a boolean context!

- Use class-based exceptions.

  String exceptions in new code are forbidden, because this language feature is being removed in Python 2.6.

  Modules or packages should define their own domain-specific base exception class, which should be subclassed from the built-in Exception class. Always include a class docstring. E.g.:

```
class MessageError(Exception):
        """Base class for errors in the email package."""
```

Class naming conventions apply here, although you should add the suffix "Error" to your exception classes, if the exception is an error. Non-error exceptions need no special suffix.

- When raising an exception, use "raise ValueError('message')" instead of the older form "raise ValueError, 'message'".

  The paren-using form is preferred because when the exception arguments are long or include string formatting, you don't need to use line continuation characters thanks to the containing parentheses. The older form will be removed in Python 3000.

- When catching exceptions, mention specific exceptions whenever possible instead of using a bare 'except:' clause.

  For example, use:

```
try:
    import platform_specific_module
except ImportError:
    platform_specific_module = None
```

  A bare 'except:' clause will catch SystemExit and KeyboardInterrupt exceptions, making it harder to interrupt a program with Control-C, and can disguise other problems. If you want to catch all exceptions that signal program errors, use 'except Exception:'.

  A good rule of thumb is to limit use of bare 'except' clauses to two cases:

  1) If the exception handler will be printing out or logging the traceback; at least the user will be aware that an error has occurred.

  2) If the code needs to do some cleanup work, but then lets the exception propagate upwards with 'raise'. 'try...finally' is a better way to handle this case.

- Additionally, for all try/except clauses, limit the 'try' clause to the absolute minimum amount of code necessary. Again, this avoids masking bugs.

  Yes:

```
try:
    value = collection[key]
```

```
        except KeyError:
            return key_not_found(key)
        else:
            return handle_value(value)
```

No:

```
        try:
            # Too broad!
            return handle_value(collection[key])
        except KeyError:
            # Will also catch KeyError raised by handle_value()
            return key_not_found(key)
```

- Use string methods instead of the string module.

  String methods are always much faster and share the same API with unicode strings. Override this rule if backward compatibility with Pythons older than 2.0 is required.

- Use ''.startswith() and ''.endswith() instead of string slicing to check for prefixes or suffixes.

  startswith() and endswith() are cleaner and less error prone. For example:

  Yes: `if foo.startswith('bar'):`

  No: `if foo[:3] == 'bar':`

  The exception is if your code must work with Python 1.5.2 (but let's hope not!).

- Object type comparisons should always use isinstance() instead of comparing types directly.

  Yes: `if isinstance(obj, int):`

  No: `if type(obj) is type(1):`

  When checking if an object is a string, keep in mind that it might be a unicode string too! In Python 2.3, str and unicode have a common base class, basestring, so you can do:

```
  if isinstance(obj, basestring):
```

  In Python 2.2, the types module has the StringTypes type defined for

that purpose, e.g.:

```
from types import StringTypes
if isinstance(obj, StringTypes):
```

In Python 2.0 and 2.1, you should do:

```
from types import StringType, UnicodeType
if isinstance(obj, StringType) or \
   isinstance(obj, UnicodeType) :
```

- For sequences, (strings, lists, tuples), use the fact that empty sequences are false.

```
Yes: if not seq:
     if seq:

No: if len(seq)
    if not len(seq)
```

- Don't write string literals that rely on significant trailing whitespace. Such trailing whitespace is visually indistinguishable and some editors (or more recently, reindent.py) will trim them.

- Don't compare boolean values to True or False using ==

```
Yes:   if greeting:

No:    if greeting == True:

Worse: if greeting is True:
```

## *References*

[1] PEP 7, Style Guide for C Code, van Rossum

[2] http://www.python.org/doc/essays/styleguide.html

[3] PEP 257, Docstring Conventions, Goodger, van Rossum

[4] http://www.wikipedia.com/wiki/CamelCase

[5] Barry's GNU Mailman style guide
http://barry.warsaw.us/software/STYLEGUIDE.txt

[6] PEP 20, The Zen of Python

[7] PEP 328, Imports: Multi-Line and Absolute/Relative

## *Copyright*

# Literaturverzeichnis

[1] Python Enhancement Proposal 8
http://www.python.org/dev/peps/pep-0008/

[2] Christoph Bühler & Patrick Lötscher, strongSwan Android 4 Client with Endpoint Assessment“, HSR Studienarbeit HS12
http://security.hsr.ch/projects/SA_2012_strongSwan-Android4-Client-with-Endpoint-Assessment.pdf

[3] Android BYOD Security based on Trusted Network Connect
http://wiki.strongswan.org/projects/strongswan/wiki/BYOD

[4] Trusted Network Connect Architecture
http://wiki.strongswan.org/projects/strongswan/wiki/TrustedNetworkConnect

[5] Trusted Computing Group IF-IMV Specification
http://www.trustedcomputinggroup.org/files/static_page_files/1D8CE1B6-1A4B-B294-D087F581D114F2DA/TNC_IFIMV_v1_3_r13.pdf

[6] Django Documentation
https://docs.djangoproject.com/en/1.5/

[7] Bootstrap Homepage
http://twitter.github.io/bootstrap/index.html

[8] jQuery Homepage
http://jquery.com

[9] Chosen Homepage
http://harvesthq.github.io/chosen/